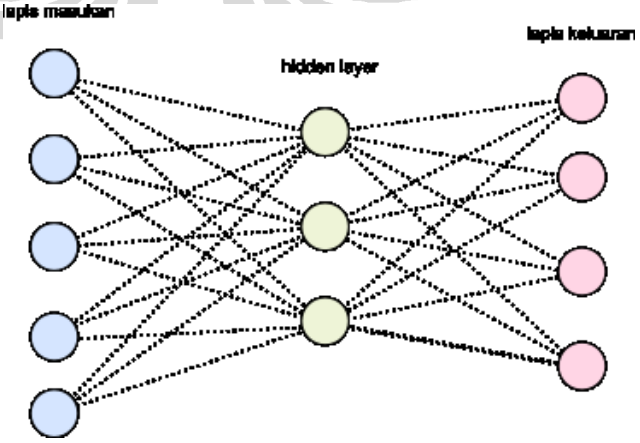
BAB 2
HEMISPHERIC STRUCTURE OF HIDDEN LAYER NEURAL NETWORK, PCA, DAN JENIS NOISE

Bab ini akan menjelaskan tentang *Hemispheric Structure Of Hidden Layer Neural Network* (HSHL-NN), *Principal Component Analysis* (PCA) dan jenis *noise*. Penjelasan HSHL-NN meliputi arsitektur, vektor posisi kamera, vektor posisi neuron, vektor pengali masukan dan keluaran *hidden layer*, fungsi aktivasi, fungsi *error* dan sinyal *error*, fungsi *error* pada pelatihan, laju pembelajaran dan momentum, aturan pelatihan HSHL-NN, algoritma HSHL-NN, serta aturan pengujian. Penjelasan PCA meliputi definisi, langkah untuk mereduksi data acuan, serta langkah untuk mereduksi data uji. Penjelasan *noise* meliputi jenis-jenis *noise* yang dipakai untuk mendegradasi citra dalam percobaan yang dilakukan pada penelitian ini.

2.1. Hemispheric structure of hidden layer neural network

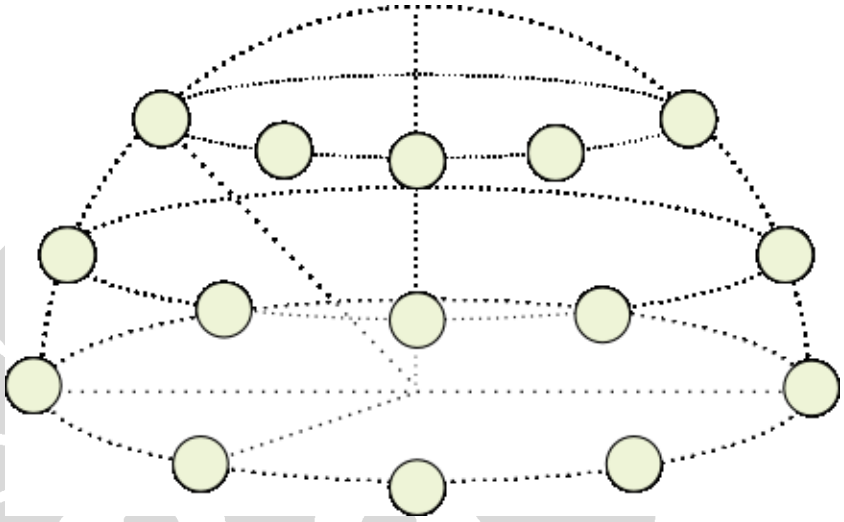
2.1.1. Arsitektur HSHL

Pada dasarnya struktur keseluruhan HSHL-NN hampir mirip dengan struktur *neural network* biasa yang terdiri dari 3 lapisan, yaitu lapis masukan, *hidden layer*, dan lapis keluaran (Gambar 2.1). Metode yang digunakan adalah dengan *backpropagation*, yang agak berbeda dengan beberapa penyesuaian.



Gambar 2. 1 Struktur *Neural Network*

Letak perbedaan HSHL-NN dengan *neural network* biasa ada pada struktur *hidden layer* yang digunakan. Neuron-neuron pada *hidden layer* ini disusun ke dalam cincin-cincin yang diameternya semakin mengecil dan ditumpuk sehingga merupai setengah bola konsentris. Setiap neuron tersebut mewakili sudut horizontal dan vertikal tertentu sesuai dengan posisinya (Widodo, 2004). Untuk lebih jelasnya lihat Gambar 2.2.



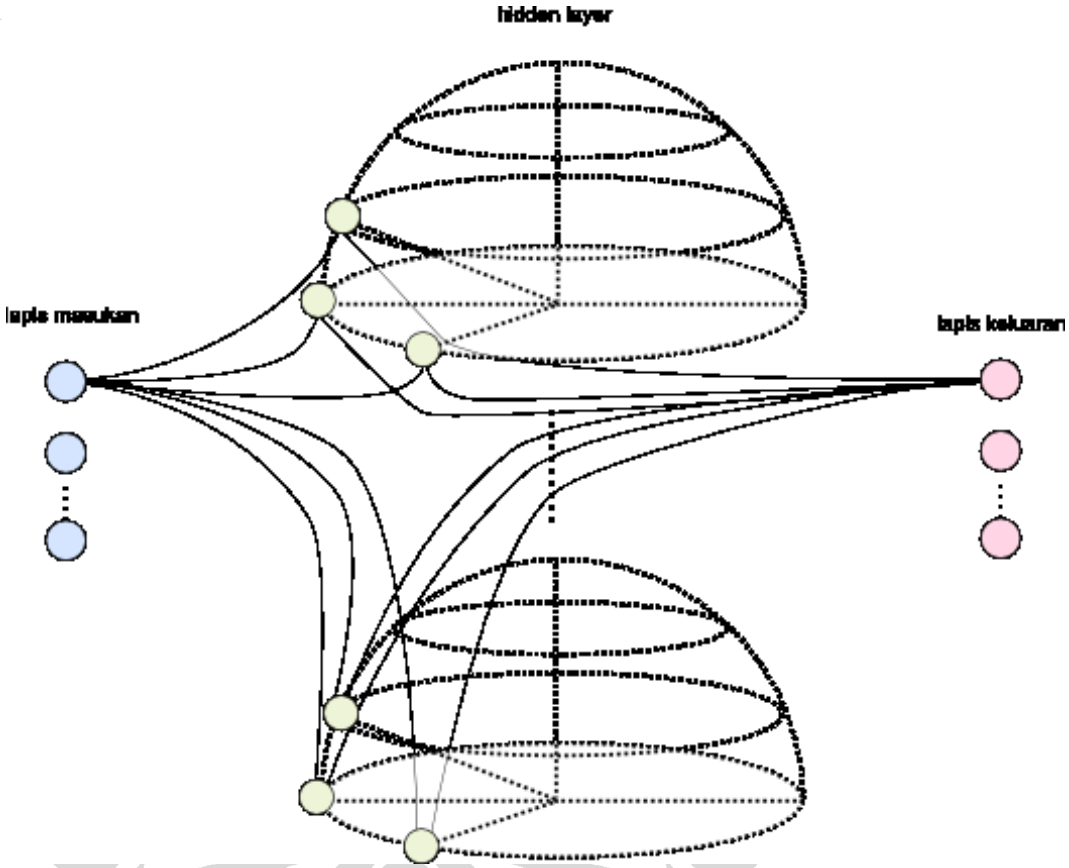
Gambar 2. 2 Struktur Neuron yang Membentuk Sebuah Hemisfer.

Lapis masukan pada HSHL-NN terdiri dari neuron-neuron yang mewakili dimensi dari data masukan. Jumlah neuron masukan pada penelitian ini adalah 50, sesuai dengan dimensi dari citra yang telah direduksi.

Hidden layer pada HSHL-NN terdiri dari beberapa jumlah hemisfer (sublapis *hidden layer*) yang jumlahnya disamakan dengan jumlah kelas objek yang ingin dikenali (Suyatno, 1999). Pada penelitian ini jumlah orang yang ingin dikenali ada empat, maka jumlah sublapis *hidden layer* ada empat. Neuron-neuron pada *hidden layer* terhubung penuh kepada setiap neuron masukan.

Lapis keluaran pada HSHL-NN terdiri dari neuron-neuron yang mewakili kelas objek yang ingin dikenali, yaitu empat buah. Sama dengan neuron-neuron pada *hidden layer* yang terhubung penuh dengan setiap neuron masukan, neuron-neuron keluaran juga terhubung penuh dengan setiap neuron pada *hidden layer*. Pada lapisan ini, setiap neuron menghasilkan nilai keluaran yang akan digunakan

untuk melakukan *update* pada bobot (pada fase pelatihan) atau menghasilkan nilai keluaran akhir (pada fase pengujian). Struktur HSHL-NN secara keseluruhan yang terdiri dari 3 lapisan dapat dilihat pada Gambar 2.3.



Gambar 2. 3 Struktur HSHL-NN

2.1.2. Vektor Posisi Kamera dan Vektor Posisi Neuron

Dalam HSHL-NN, untuk merepresentasikan sudut pandang objek dalam ruang tiga dimensi digunakan koordinat bola. Pusat objek dianggap berada di titik pusat hemisfer dan kamera atau neuron berada pada titik-titik pada kulit hemisfer. Dengan demikian, dapat direpresentasikan sudut pandang kamera/neuron terhadap objek dengan koordinat bola. Transformasi koordinat bola dari posisi kamera ke dalam koordinat kartesius akan menghasilkan vektor posisi kamera. Lalu transformasi koordinat bola dari posisi neuron ke dalam koordinat Kartesius akan menghasilkan vektor posisi neuron.

Vektor Posisi Kamera

Setiap titik posisi dari kamera direpresentasikan dengan koordinat bola (ρ, θ, φ) dimana ρ adalah panjang OK, θ adalah sudut antara K' dengan sumbu X positif, dan φ adalah sudut antara vektor $|OK|$ dengan sumbu Z positif. Karena ρ adalah jari-jari bola $R_{d(k)}$, maka vektor posisi kamera $\boldsymbol{d}(k)$ adalah

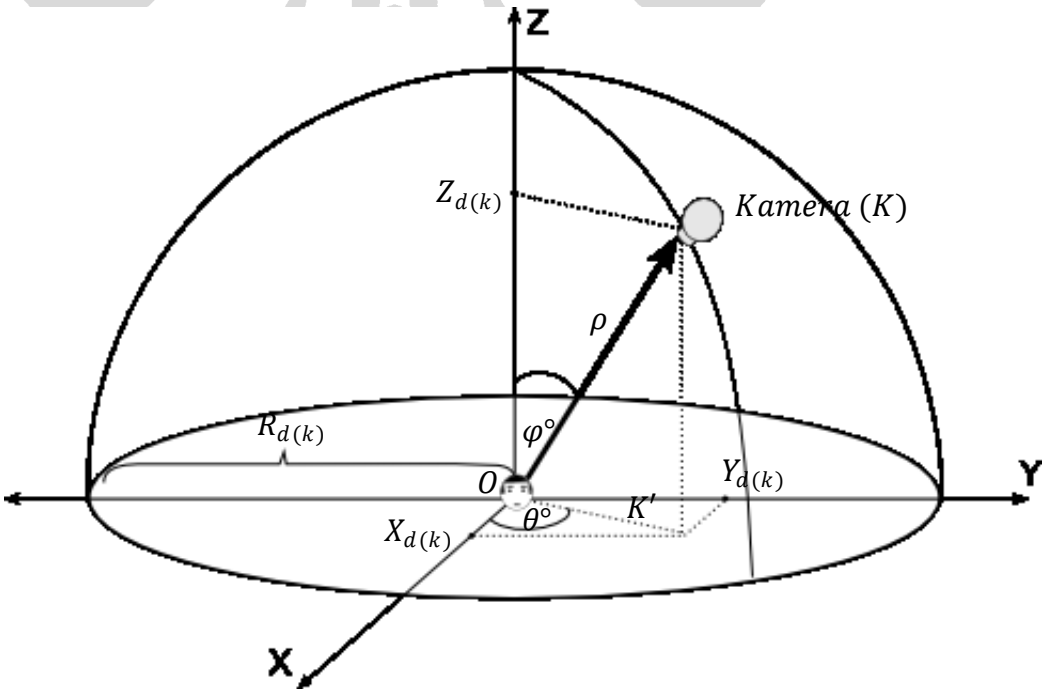
$$\boldsymbol{d}(k) = \left[R_{d(k)}, \theta_{d(k)}, \varphi_{d(k)}\right] \quad (2\text{-}1)$$

Transformasi dari koordinat bola ke koordinat Kartesius dilakukan dengan pemanfaatan sifat trigonometri.

$$\boldsymbol{d}(k) = \left[X_{d(k)}, Y_{d(k)}, Z_{d(k)}\right] \quad (2\text{-}2)$$

$$\boldsymbol{d}(k) = \left[R_{d(k)} sin\theta_{d(k)} cos\varphi_{d(k)}, R_{d(k)} sin\theta_{d(k)} sin\varphi_{d(k)}, R_{d(k)} cos\varphi_{d(k)}\right] \quad (2\text{-}3)$$

Representasi secara geometris dapat dilihat pada Gambar 2.4. Dalam tulisan ini, Jari-jari bola $R_{d(k)}$, dianggap sama dengan 1.



Gambar 2. 4 Vektor Posisi Kamera (Panah Tebal)

Vektor Posisi Neuron

Seperti kamera, setiap neuron pada *hidden layer* memiliki representasi koordinat bola (ρ, θ, φ) dengan ρ adalah panjang ON, θ adalah sudut antara N' dengan sumbu X positif, dan φ adalah sudut antara vektor $|ON|$ dengan sumbu Z positif. Karena ρ adalah jari-jari bola $R_{v(st)}$, maka vektor posisi neuron $\boldsymbol{v}(st)$ adalah:

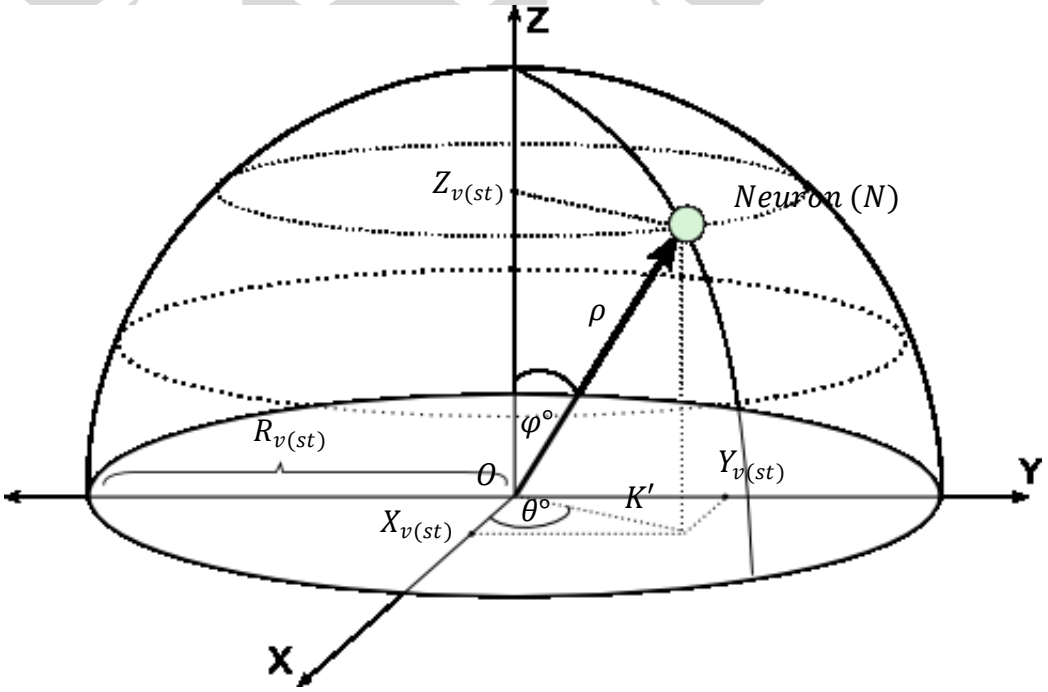
$$\boldsymbol{v}(st) = \left[R_{v(st)}, \theta_{v(st)}, \varphi_{v(st)}\right] \tag{2-4}$$

Transformasi dari koordinat bola ke koordinat Kartesius dilakukan dengan pemanfaatan sifat trigonometri:

$$\boldsymbol{v}(st) = \left[X_{v(st)}, Y_{v(st)}, Z_{v(st)}\right] \tag{2-5}$$

$$\boldsymbol{v}(st) = \left[R_{v(st)} sin\theta_{v(st)} cos\varphi_{v(st)}, R_{v(st)} sin\theta_{v(st)} sin\varphi_{v(st)}, R_{v(st)} cos\varphi_{v(st)}\right] \tag{2-6}$$

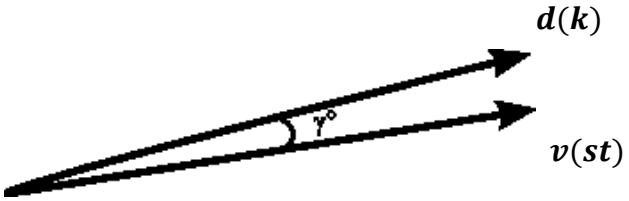
Representasi secara geometris dapat dilihat pada Gambar 2.5.Subskrip s adalah indeks neuron pada suatu cincin di hemisfer.Subskrip t adalah indeks cincin pada hemisfer.



Gambar 2. 5 Vektor Posisi Neuron (Panah Tebal)

2.1.3. Faktor Pengali Masukan dan Keluaran *Hidden Layer*

Vektor posisi kamera dan vektor posisi neuron yang telah dijelaskan sebelumnya digunakan untuk menentukan faktor pengali masukan gh_{st} dan faktor keluaran go_{st} pada *hidden layer*.



Gambar 2. 6 Vektor Posisi Kamera Dihimpitkan dengan Vektor Posisi Neuron

Bila titik pusat vektor posisi kamera dan vektor posisi neuron dihimpitkan (Gambar 2.6) maka perbedaan posisi kamera dengan posisi neuron dapat direpresentasikan sebagai *dot product* antara $\boldsymbol{d}(k)$ dan $\boldsymbol{v}(st)$:

$$\boldsymbol{d}(k) \bullet \boldsymbol{v}(st) = \|\boldsymbol{d}(k)\| \, \|\boldsymbol{v}(st)\| cos\, \gamma \tag{2-7}$$

Keterangan:

- γ adalah selisih sudut antara vektor posisi kamera dengan vektor posisi neuron, $0 < \gamma < \pi$
- $$\|\boldsymbol{d}(k)\| = \sqrt{X_{\boldsymbol{d}(k)}{}^2 + Y_{\boldsymbol{d}(k)}{}^2 + Z_{\boldsymbol{d}(k)}{}^2} \tag{2-8}$$
- $$\|\boldsymbol{v}(st)\| = \sqrt{X_{\boldsymbol{v}(st)}{}^2 + Y_{\boldsymbol{v}(st)}{}^2 + Z_{\boldsymbol{v}(st)}{}^2} \tag{2-9}$$

Setelah mendapatkan *dot product* dari $\boldsymbol{d}(k)$ dan $\boldsymbol{v}(st)$ dapat ditentukan faktor masukan *hidden layer* gh_{st} sebagai berikut:

$$gh_{st} = \begin{cases} \boldsymbol{d}(k) \bullet \boldsymbol{v}(st) & jika\ \boldsymbol{d}(k) \bullet \boldsymbol{v}(st) \geq 0 \\ 0 & jika\ \boldsymbol{d}(k) \bullet \boldsymbol{v}(st) < 0 \end{cases} \tag{2-10}$$

Sedangkan faktor keluaran *hidden layer* go_{st} sebagai berikut:

$$go_{st} = \begin{cases} 1 & jika\ \boldsymbol{d}(k) \bullet \boldsymbol{v}(st) \geq 0 \\ 0 & jika\ \boldsymbol{d}(k) \bullet \boldsymbol{v}(st) < 0 \end{cases} \tag{2-11}$$

Dengan menggunakan persamaan (2-10) dan (2-11), nilai masukan ke setiap neuron pada *hidden layer* dari lapis masukan adalah sebagai berikut

$$Z_{st} = S(z\_in_{st}) \tag{2-12}$$

$$z\_in_{st} = gh_{st}(b_{jst} + \sum_{i=1}^{I}\sum_{j=1}^{J} x_i v_{ijst}) \tag{2-13}$$

Sedangkan untuk nilai keluaran dari *hidden layer* menuju lapis keluaran adalah sebagai berikut:

$$O_k = S(y\_in_k) \tag{2-14}$$

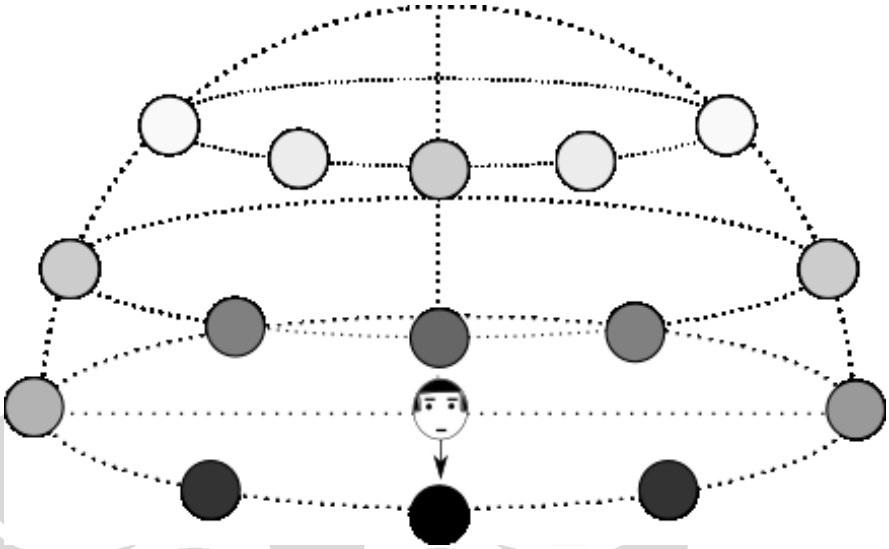
$$y\_in_k = b_k \sum_{j=1}^{J}\sum_{s=1}^{S}\sum_{t=1}^{T} go_{st} w_{kjst} z_{jst} \tag{2-15}$$

Dapat terlihat dari persamaan (2-12) bahwa faktor masukan gh_{st} mempengaruhi besarnya nilai masukan pada setiap neuron pada *hidden layer* dalam proses pelatihan. Jika neuron memiliki vektor posisi neuron searah dengan vektor posisi kamera dari kamera akan mendapatkan sinyal masukan yang lebih besar, sedangkan semakin besar sudut γ antara vektor posisi neuron dengan vektor posisi kamera maka akan semakin kecil pula sinyal masukan yang diterima. Untuk sudut dengan perbedaan cukup besar/vektornya berseberangan (memiliki sudut $\gamma < -\frac{\pi}{2}$ atau $\gamma > \frac{\pi}{2}$) (Widodo, 2004), maka nilai masukan tidak diperhitungkan atau 0.

Faktor keluaran go_{st} digunakan untuk menentukan nilai yang akan dikirim ke lapis keluaran. Hampir serupa dengan faktor masukan gh_{st}, faktor keluaran ini digunakan untuk memberikan sinyal yang lebih besar kepada neuron yang harus banyak belajar (Widodo, 2004). Dapat terlihat pada persamaan (2-13), jika nilai go_{st} 1 maka nilai keluaran dari *hidden layer* akan mempengaruhi masukan kepada lapis keluaran, sebaliknya jika nilai go_{st} adalah 0 maka diabaikan.

Pada gambar 2.7 dapat terlihat intensitas masukan yang diterima neuron pada *hidden layer* jika ada citra wajah dengan sudut pandang kamera $\theta = 0°, \varphi = 0°$. Tingkat kegelapan warna pada setiap neuron merepresentasikan intensitas

masukan neuron tersebut. Dengan kata lain, semakin gelap warna neuron, semakin besar pula masukan yang akan diterima, sebaliknya semakin terang warna dari neuron maka semakin kecil masukan yang akan diterima. Begitu seterusnya sesuai dengan sudut pandang citra wajah yang ada.



Gambar 2. 7 Intensitas Masukan Tiap Neuron pada *hidden layer* dengan Sudut $\theta = 0°, \varphi = 0°$

2.1.4. Inisalisasi Bobot dan Bias

Pada pelatihan *neural network*, inisialisasi bobot dan bias merupakan salah satu faktor yang cukup penting. Bobot awal merupakan posisi di ruang *error* pada saat awal pelatihan. Hal ini menjadi penting karena menentukan proses pelatihan akan mencapai titik minimum global atau minimum lokal, serta mempengaruhi kecepatan konvergensi jaringan.

Apabila distribusi dari nilai bobot terlalu besar, maka neuron-neuron pada *hidden layer* akan cepat mencapai nilai jenuhnya pada *epoch-epoch* awal. Metode dalam melakukan inisialisasi bobot yang ideal adalah yang menghasilkan pembelajaran yang cepat dan seragam, artinya secara bersamaan semua bobot dapat meraih nilai *equilibrium* yang tepat (Widodo, 2004).

Metode yang umum digunakan adalah inisialisasi bobot secara *random* (acak). Dalam penelitian ini, metode inisialisasi yang digunakan adalah Nguyen-Widrow, dengan aturan sebagai berikut:

1. Inisialisasi bobot antara *hidden layer* dengan lapis keluaran w_{qst} dan bias lapis keluaran b_q dengan nilai acak n, dimana $-0.5 < n < 0.5$.

2. Inisialisasi bobot antara lapis masukan dengan *hidden layer* dengan nilai acak n, dimana $-0.5 < n < 0.5$.

3. Hitung faktor skala β:

$$\beta = 0.7\rho^{1/n} \tag{2-16}$$

Keterangan:

- ρ jumlah neuron pada *hidden layer*
- n jumlah neuron pada lapis masukan

4. Hitung *norm* bobot v_{st}:

$$\|v_{st}\| = \sqrt{\sum_{s=1}\sum_{t=1} {v_{st}}^2} \tag{2-17}$$

5. Perbaiki bobot awal:

$$v_{ijst} = \frac{\beta\, v_{ijst}}{\|v_{st}\|} \tag{2-18}$$

6. Inisialisasi bias neuron *hidden layer* dengan nilai antara $[-\beta, \beta]$.

2.1.5. Fungsi aktivasi

Fungsi aktivasi $\boldsymbol{S}(\bullet)$ digunakan untuk memetakan nilai masukan pada suatu neuron menjadi nilai keluaran. Syarat dari fungsi aktivasi yang baik adalah harus efisien dalam perhitungannya dan menunjang proses pelatihan. Beberapa hal yang

perlu diperhatikan dalam melakukan pemilihan fungsi aktivasi adalah (Duda, 2000):

- $S(\bullet)$ adalah fungsi non-linear.
- $S(\bullet)$ memiliki nilai asimtotik untuk masukan yang semakin besar atau semakin kecil. Dengan kata lain S(•) memiliki nilai jenuh. Hal ini diperlukan untuk mengkompresi nilai masukan kedalam range tertutup.
- $S(\bullet)$ kontinyu dan *differentiable* untuk semua nilai •. Hal ini penting karena metode pembelajaran propagasi balik menggunakan turunan dari fungsi aktivasi untuk menentukan perbaikan bobot. $S(\bullet)$ naik secara monoton dan dari segi biaya komputasi, lebih baik lagi jika $S'(\bullet)$ dapat dinyatakan dalam $S(\bullet)$.

Fungsi yang memiliki karakteristik yang telah disebutkan sebelumnya adalah fungsi *sigmoid*. Fungsi ini ada dua jenis yaitu fungsi *sigmoid biner* dan fungsi *sigmoid bipolar*.

- Fungsi *sigmoid biner*

$$f(x) = \frac{1}{1+\exp(-x)} \tag{2-19}$$

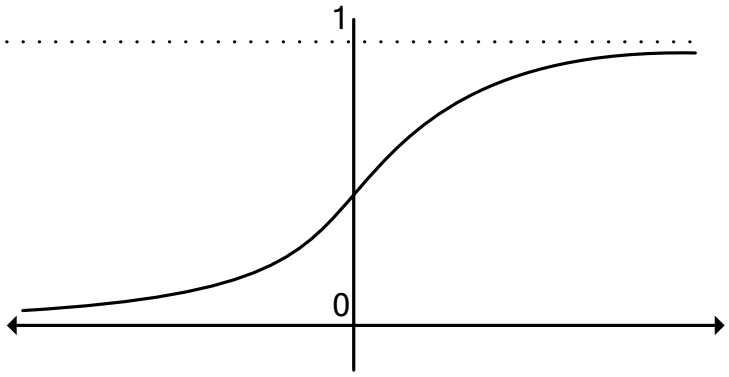
$$f'(x) = f(x)[1 - f(x)] \tag{2-20}$$

- Fungsi *sigmoid bipolar*

$$f(x) = \frac{2}{1+\exp(-x)} - 1 \tag{2-21}$$

$$f'(x) = \frac{1}{2}[1 + f(x)][1 - f(x)] \tag{2-22}$$

Pada penelitian ini, fungsi aktivasi yang digunakan adalah fungsi *sigmoid biner*. Plot fungsi *sigmoid biner* dapat dilihat pada Gambar 2.8.



Gambar 2. 8 Fungsi *Sigmoid Biner*

2.1.6. Fungsi *Error* dan Sinyal *Error*

Fungsi *error* dan sinyal *error* digunakan untuk memberikan umpan balik pada proses pelatihan HSHL-NN. Metode pembelajaran yang dilakukan menggunakan pendekatan *gradient descent* dalam penentuan sinyal *error*. Arah dan besarnya gradien dapat diturunkan dengan menggunakan fungsi aktivasi pada neuron lapis keluaran.

Pada penelitian ini, fungsi *error* yang dipakai adalah fungsi *error cross entropy*. Fungsi tersebut adalah sebagai berikut

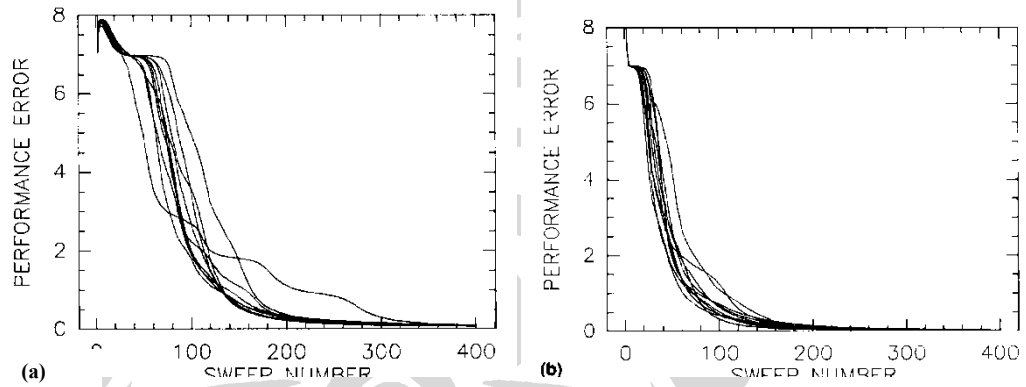
$$E = \sum -t_q \ln(o_q) - (1 - t_q)\ln(1 - o_q) \quad (2\text{-}23)$$

Alasan digunakannya fungsi *error cross entropy* karena kemampuannya untuk dapat mempercepat konvergensi jaringan (Ooyen, Arjen, dan Nienhuis, 1992). Pada fungsi *error Root Mean Squared Error* neuron keluaran dapat memberikan respon yang sangat jauh dari target, tetapi menghasilkan sinyal *error* yang kecil. Penggunaan *cross entropy* sebagai fungsi kesalahan menyebabkan sinyal *error* berbanding lurus dengan beda antara nilai keluaran (o_q) dan target (t_q) sehingga sinyal *error* tetap cukup besar untuk *error* yang ekstrim. Hal ini dapat mempercepat konvergensi pada epoch awal, saat bobot masih sedikit menyesuaikan diri.

Pembuktian bahwa fungsi *error cross entropy* lebih cepat mencapai konvergensi dalam pembelajaran dibandingkan fungsi *error Root Mean Squared Error* telah

dilakukan pada penelitian sebelumnya, berikut adalah hasil percobaan yang dilakukan

Gambar 2.9 menunjukkan perbandingan antara kedua metode fungsi *error*. *Neural network* yang digunakan terdiri atas delapan neuron masukan, tiga *hidden* neuron dan delapan neuron keluaran. *Neural network* tersebut digunakan untuk menyelesaikan masalah *encoding*. Jaringan dilatih untuk memetakan pola masukan berupa *string* 8 bit yang saling ortogonal, yang dipetakan menjadi *string* 3 bit pada neuron tersembunyi, dan kemudian dipetakan kembali menjadi *string* 8 bit yang sama dengan pola masukan di neuron keluaran. Dapat dilihat konvergensi pembelajaran dengan *cross entropy* lebih cepat dibandingkan dengan *root mean squared error* (Widodo, 2004).



Gambar 2. 9 Perbandingan Kecepatan Konvergensi antara Pembelajaran dengan (a) *mean squared error* dan (b) *cross entropy*

Walaupun menguntungkan, penurunan *error* dengan terlalu cepat dapat menyebabkan kondisi *overfitting* dimana neural network kehilangan kemampuan untuk menggeneralisasi masukan yang tidak dilatih. Untuk menghindari hal tersebut, jumlah *epoch* pelatihan pada penelitian ini dibatasi apabila sudah mencapai tingkat *error* pelatihan tertentu (Widodo, 2004). Mengenai tingkat *error* pelatihan akan dibahas pada bagian berikutnya.

2.1.7. Fungsi *Error* Pelatihan

Fungsi *error* yang telah dijelaskan sebelumnya digunakan dalam menghitung umpan balik bagi jaringan untuk setiap pola masukan pelatihan yang diterima.

Dalam satu *epoch*, jaringan menerima beberapa pola masukan pelatihan. Fungsi *error* dihitung untuk setiap masukan tersebut. Untuk memonitor laju pembelajaran secara menyeluruh, dilakukan pengakumulasian nilai fungsi *error* dari setiap *epoch* selama pelatihan berlangsung menggunakan fungsi *error* pelatihan (*training error function*).

Fungsi *error* pelatihan yang digunakan pada penelitian ini adalah fungsi *root mean square normalized error,* dengan persamaan sebagai berikut

$$E_{rms} = \frac{1}{PQ}\sqrt{\sum_{p=1}^{P}\sum_{q=1}^{Q}(t_{pq} - o_{pq})^2} \tag{2-24}$$

Keterangan:

- P adalah jumlah pola masukan yang dilatih.
- Q adalah jumlah neuron pada lapis keluaran

2.1.8. Laju Pembelajaran dan Momentum

Untuk melakukan perbaikan pada bobot dalam jaringan, digunakan aturan yang sama seperti yang digunakan pada *backpropagation* sebagai berikut:

$$w_{k+1} = w_k + \Delta w_k \tag{2-25}$$

$$\Delta w_k = \alpha\delta_k + \eta\Delta w_{k-1} \tag{2-26}$$

Keterangan:

- α adalah laju pembelajaran, $0 < \alpha < 1$.
- η adalah momentum. $0 < \eta < 1$.
- δ_k adalah nilai *error* dari neuron pada lapis keluaran maupun *error* dari *hidden layer*.

Momentum dikenalkan oleh Rumelhart pada 1986. Momentum digunakan dengan tujuan untuk mempertahankan stabilitas jaringan. Momentum yang besar menjaga agar perubahan bobot selalu ke arah yang global minimum. Jika dianalogikan seperti sebuah mobil biasa dengan truk yang besar, walaupun keduanya berjalan dengan kecepatan sama dan menuju arah yang sama, truk tidak mudah untuk mengubah arahnya dibandingkan dengan mobil biasa (Widodo, 2004).

2.1.9. Aturan Pembelajaran HSHL

Sama seperti *neural network* with *backpropagation,* aturan pembelajaran HSHL-NN terdiri dari dua tahapan, yaitu *feedforward* dan *backpropagation.* Pada fase *feedforward* setiap neuron masukan x_{ij} memberikan nilai masukan ke setiap neuron v_{jst} pada *hidden layer*, kemudian setiap neuron pada *hidden layer* tersebut memberikan nilai masukan kepada neuron y_k pada lapis keluaran yang akan mengeluarkan nilai keluaran o_k. Sesuai dengan pendekatan *backpropagation* yang pelatihannya *supervised*, maka nilai keluaran o_k dari setiap neuron pada lapis akhir akan dibandingkan dengan nilai yang seharusnya t_k dan akan dilakukan kalkulasi pada fase *backpropagation.*

$$\delta_k = o_k - t_k \tag{2-27}$$

Pada fase *backpropagation* semua bobot dari jaringan, baik bobot antara lapis masukan dengan *hidden layer* maupun bobot antara *hidden layer* dengan lapis keluaran, akan disesuaikan dengan aturan sebagai berikut

- Dalam proses *update* bobot w_{kjst}, jaringan antara *hidden layer* dan lapis keluaran

$$w_{kjst}\,(t+1) = w_{kjst}\,(t) + \Delta w_{kjst}\,(t+1) \tag{2-28}$$

$$\Delta w_{kjst}\,(t+1) = \alpha\delta_k z_{jst}\, go_{st} + \eta\Delta w_{kj\,st}\,(t) \tag{2-29}$$

- *Update* bias b_k pada neuron *hidden layer* dan lapis keluaran adalah

$$b_k(t+1) = b_k(t) + \Delta b_k(t+1) \tag{2-30}$$

$$\Delta b_k(t+1) = \alpha\delta_k + \eta\Delta b_k(t) \tag{2-31}$$

- Dalam proses *update* bobot v_{ijst} pada jaringan antara lapis masukan dan *hidden layer* juga menggunakan persamaan yang sama dengan persamaan (2-26) namun harus mencari nilai sinyal *error* δ_{jst} yang dihitung dengan:

$$\delta_{jst} = \sum \delta_{jst}\, w_{kst}\, f'(Z\_in_{jst})(1 - z_{jst}) \quad (2\text{-}32)$$

$$v_{ijst}(t+1) = v_{ijst}(t) + \Delta v_{ijst}(t+1) \quad (2\text{-}33)$$

$$\Delta v_{ijst}(t+1) = \alpha \delta_{jst}\, x_i\, gh_{st} + \eta \Delta v_{ijst}(t) \quad (2\text{-}34)$$

- *Update* bias b_{st} pada neuron *hidden layer* adalah
  - $b_{st}(t+1) = b_{st}(t) + \Delta b_{st}(t+1)$ (2-35)
  - $\Delta b_{st}(t+1) = \alpha \delta_{st} + \eta \Delta b_{st}(t)$ (2-36)

2.1.10. Algoritma HSHL-NN

Algoritma HSHL-NN terbagi ke dalam dua bagian, yaitu dalam melakukan proses pelatihan dan pengujian.

Algoritma Pelatihan

Sebelumnya telah dibahas mengenai dua tahap yang dilakukan pada pelatihan adalah fase *feedforward* dan *backpropagation*. Algoritma yang digunakan adalah sebagai berikut

1. Inisialisasi bobot awal pada setiap jaringan (antara lapis masukan-*hidden layer* dan *hidden layer*-lapis keluaran), bias tiap neuron pada *hidden layer* dan lapis keluaran, laju pembelajaran α, momentum η, batas *epoch*, dan batas tingkat *error* pelatihan.
2. Jika batas tingkat *error* belum tercapai, lakukan lagi langkah 3-4.
3. Fase *feedforward*
   a. Hitung nilai gh_{st} pada setiap neuron *hidden layer* (2-10).

b. Hitung nilai go_{st} pada setiap neuron *hidden layer* (2-11).

c. Hitung nilai masukan z_in_{st} setiap neuron pada lapisan tersembunyi (2-13).

d. Hitung nilai aktivasi z_{st} setiap neuron pada *hidden layer* (2-12).

e. Hitung nilai masukan y_in_k setiap neuron pada lapisan keluaran (2-15).

f. Hitung nilai aktivasi o_k setiap neuron pada lapis keluaran (2-14).

4. Fase *backpropagation*

a. Bandingkan nilai aktivasi o_k neuron lapis keluaran dengan target t_k yang telah ditentukan dengan menghitung sinyal *error* (2-27).

b. *Update* bobot Δw_{kjst} pada jaringan antara *hidden layer* dengan lapis keluaran (2-29) dan *update* bias Δb_k neuron pada lapis keluaran (2-31).

c. Hitung nilai sinyal *error* δ_{jst} setiap neuron pada *hidden layer* (2-32).

d. *Update* bobot Δv_{ijst} pada jaringan antara lapis masukan dengan *hidden layer* (2-34) dan *update* bias Δb_{st} neuron pada *hidden layer* (2-36).

e. Ubah bobot w_{kjst} pada jaringan antara *hidden layer* dengan lapis keluaran (2-28) dan bias pada neuron di lapis keluaran (2-30).

f. Ubah bobot v_{ijst} pada jaringan antara lapis masukan dengan *hidden layer* (2-33) dan bias pada neuron di *hidden layer* (2-35).

5. Hitung total *error epoch* dengan menggunakan fungsi *root mean square normalized error* (2-24).

6. Lakukan komparasi nilai total *error epoch* dengan batas tingkat *error*, jika nilai total *error* masih lebih besar maka lakukan langkah 3 - 4 kembali, jika tidak maka hentikan proses pelatihan (pelatihan selesai).

Algoritma Pengujian

Pada proses pengujian, algoritma yang digunakan hampir sama dengan proses pelatihan, letak perbedaannya yaitu proses pengujian dilakukan hanya sampai fase *feedforward* saja. Berikut ini adalah penjabarannya:

1. Inisialisasi bobot awal pada setiap jaringan (antara lapis masukan-*hidden layer* dan *hidden layer*-lapis keluaran), bias tiap neuron pada *hidden layer* dan lapis keluaran, laju pembelajaran α, momentum η, batas *epoch*, dan batas tingkat *error* pelatihan.

2. Untuk setiap skema pengujian lakukan langkah 3-8.

3. Hitung nilai gh_{st} pada setiap neuron *hidden layer* (2-10).

4. Hitung nilai go_{st} pada setiap neuron *hidden layer* (2-11).

5. Hitung nilai masukan z_in_{st} setiap neuron pada *hidden layer* (2-13).

6. Hitung nilai aktivasi z_{st} setiap neuron pada *hidden layer* (2-12).

7. Hitung nilai masukan y_in_k setiap neuron pada lapis keluaran (2-15).

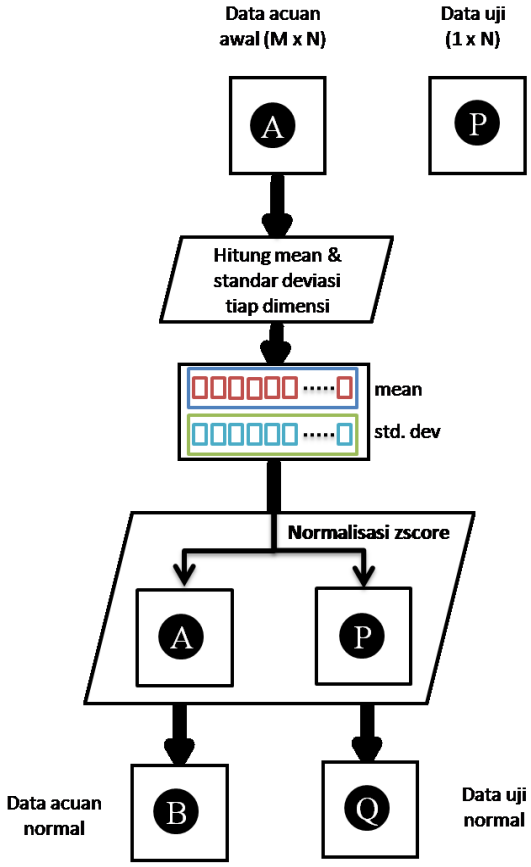
8. Hitung nilai aktivasi o_k setiap neuron pada lapis keluaran (2-14).

Nilai o_k pada setiap neuron lapis keluaran inilah yang digunakan untuk menentukan apakah hasil keluaran HSHL-NN benar atau tidak.

2.2. *Principal Component Analysis* (PCA)

2.2.1. Definisi

Tidak jarang penelitian, terutama dalam bidang pengenalan wajah menggunakan data yang memiliki jumlah dimensi yang sangat besar. Sebagai contoh, jika data yang dipakai sejumlah M citra wajah dengan jumlah dimensi N, maka besarnya data adalah M x N. Jika nilai N cukup besar maka hal ini dapat membuat *cost* dalam komputasi menjadi besar pula dan kadang hasil yang didapatkan tidak terlalu baik. Untuk mengatasinya dapat digunakan sebuah teknik yang bernama *Principal Component Analysis* (PCA). Dengan PCA kita dapat mereduksi data menjadi M x *n* dimana *n* < N.

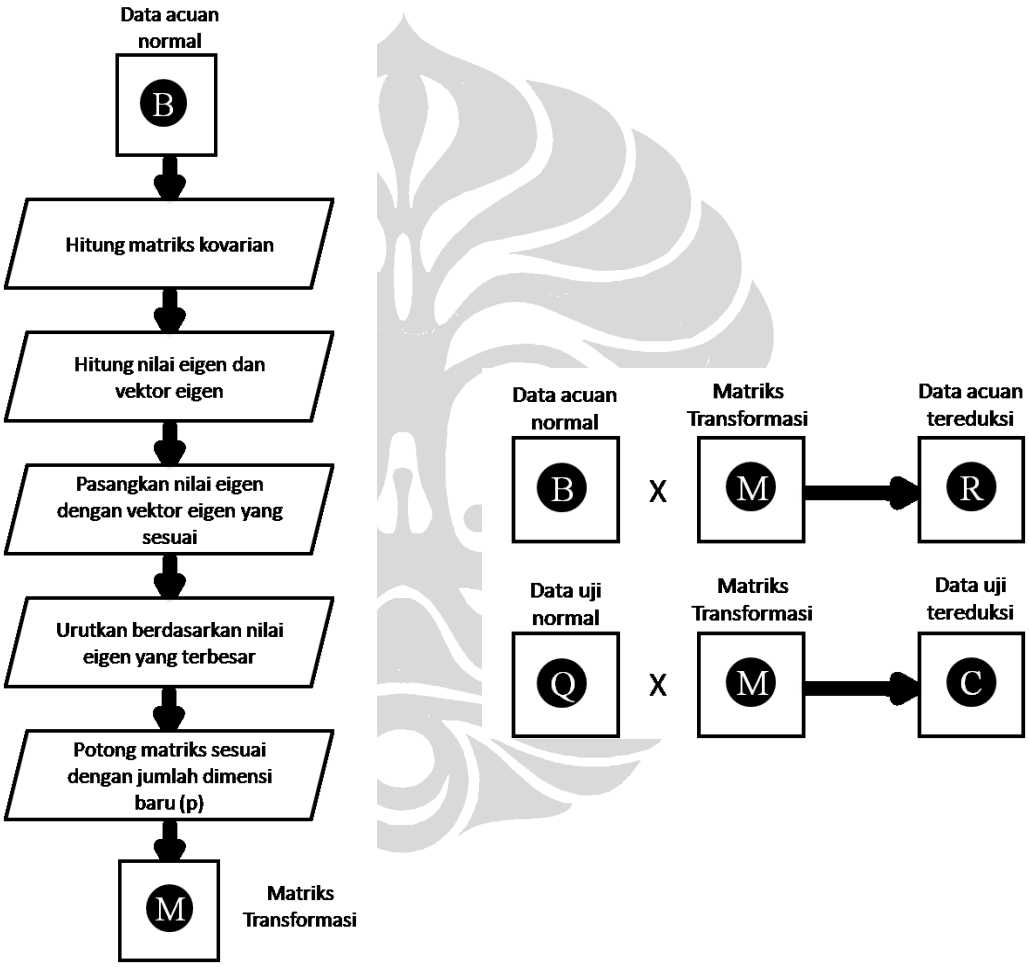


Gambar 2. 10 Alur Proses PCA (1)

Principal Component Analysis (PCA) adalah sebuah teknik untuk mengidentifikasi pola dalam data, dan menyajikan data tersebut sehingga terlihat dengan jelas persamaan dan perbedaannya. Dengan mengetahui pola tersebut, kita

dapat melakukan kompresi data (dalam hal ini citra wajah) dengan mengurangi jumlah dimensi tanpa menghilangkan informasi penting dari data.

Pada Gambar 2.10 dan Gambar 2.11 dapat terlihat alur keseluruhan dari proses PCA yang digunakan dalam proses reduksi dimensi citra wajah yang digunakan pada penelitian ini. Untuk memudahkan pemahaman, maka proses PCA akan dijelaskan secara langkah demi langkah. Langkah-langkah tersebut terbagi atas dua bagian, yaitu PCA untuk menentukan data acuan baru dan data uji baru.



Gambar 2. 11 Alur Proses PCA (2)

2.2.2. Reduksi Data Acuan

Berikut ini adalah langkah untuk mereduksi data acuan:

1. **Susun data yang akan digunakan**

Representasikan citra wajah sebagai sebuah vektor M, misalkan sebuah citra dengan ukuran *n* x *n* piksel maka vektornya adalah $M = (m_1, m_2, \dots, m_{n^2})$. Jika terdapat *k* buah citra maka bentuk matriks dari kumpulan vektor-vektor wajah sebagai berikut:

$$ImgMat = \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_k \end{bmatrix} \tag{2-37}$$

$$ImgMat = \begin{bmatrix} m_{1,1} & m_{1,2} & \dots & m_{1,n^2} \\ m_{2,1} & m_{2,2} & \dots & m_{2,n^2} \\ \vdots & \vdots & \ddots & \vdots \\ m_{k,1} & m_{k,2} & \dots & m_{k,n^2} \end{bmatrix} \tag{2-38}$$

2. Hitung rata-rata dan standar deviasi tiap dimensi

Untuk tiap dimensi, hitung rata-rata dan standar deviasinya. Kemudian bentuk vektor *mean* dan vektor standar deviasi.

$$\begin{bmatrix} m_{1,1} & m_{1,2} & \dots & m_{1,n^2} \\ m_{2,1} & m_{2,2} & \dots & m_{2,n^2} \\ \vdots & \vdots & \ddots & \vdots \\ m_{k,1} & m_{k,2} & \dots & m_{k,n^2} \end{bmatrix} \tag{2-39}$$

$$\bar{x}_i = \frac{1}{k} . \sum_{j=1}^{k} m_{i,j} \; 0 < i < n^2 \tag{2-40}$$

$$\sigma_i = \sqrt{\frac{1}{k} \sum_{j=1}^{k} \left(m_{i,j} - \bar{x}_i\right)^2} \tag{2-41}$$

$$meanVec = (\bar{x}_1, \bar{x}_2, \dots, \bar{x}_{n^2}) \tag{2-42}$$

$$stdVec = (\sigma_1, \sigma_2, \dots, \sigma_{n^2}) \tag{2-43}$$

3. Lakukan normalisasi dengan zscore

Hitung normalisasi untuk tiap elemen matriks, akan terbentuk matriks dengan data yang ternormalisasi.

$$z_{i,j} = \frac{m_{i,j} - \bar{x}_j}{\sigma_j} \tag{2-44}$$

$$ImgMatNorm = \begin{bmatrix} z_{1,1} & z_{1,2} & \dots & z_{1,n^2} \\ z_{2,1} & z_{2,2} & \dots & z_{2,n^2} \\ \vdots & \vdots & \ddots & \vdots \\ z_{k,1} & z_{k,2} & \dots & z_{k,n^2} \end{bmatrix} \tag{2-45}$$

4. Hitung matriks kovarian

$$cov\,(X,Y) = \frac{\sum_{i=1}^{k}(X_i - \bar{X})(Y_i - \bar{Y})}{(k-1)} \tag{2-46}$$

$$C^{nxn} = \left(c_{i,j}, c_{i,j} = cov(Dim_i, Dim_j)\right) \tag{2-47}$$

$$covMat = \begin{bmatrix} cov(D_1, D_1) & cov(D_1, D_2) & \dots & cov(D_1, D_{n^2}) \\ cov(D_2, D_1) & cov(D_2, D_2) & \dots & cov(D_2, D_{n^2}) \\ \vdots & \vdots & \ddots & \vdots \\ cov(D_{n^2}, D_1) & cov(D_{n^2}, D_2) & \dots & cov(D_{n^2}, D_{n^2}) \end{bmatrix} \tag{2-48}$$

5. Hitung nilai Eigen dan vektor Eigen dari matriks kovarian

Nilai Eigen ditentukan dengan persamaan berikut:

$$\det\begin{bmatrix} c_{1,1} - \lambda & c_{1,2} & \dots & c_{1,n^2} \\ c_{2,1} & c_{2,2} - \lambda & \dots & c_{2,n^2} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n^2,1} & c_{n^2,2} & \dots & c_{n^2,n^2} - \lambda \end{bmatrix} = 0 \tag{2-49}$$

$$\begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_{n^2} \end{bmatrix} \tag{2-50}$$

Kemudian susun nilai Eigen ke dalam sebuah vektor kolom

$$eigVal = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_{n^2} \end{bmatrix} \tag{2-51}$$

Cari vektor Eigen

$$\begin{bmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,n^2} \\ c_{2,1} & c_{2,2} & \dots & c_{2,n^2} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n^2,1} & c_{n^2,2} & \dots & c_{n^2,n^2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n^2} \end{bmatrix} = \lambda_i \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n^2} \end{bmatrix}, eig_i = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n^2} \end{bmatrix} \quad (2\text{-}52)$$

$$eigMat = (eig_1, eig_2, \dots, eig_{n^2}) \quad (2\text{-}53)$$

Kumpulan dari vektor Eigen yang berupa vektor kolom akan membentuk sebuah matriks. Matriks ini kemudian ditranspose dan disisipkan vektor nilai Eigen pada kolom pertama dan urutkan baris matriks sesuai dengan nilai Eigen yang terbesar.

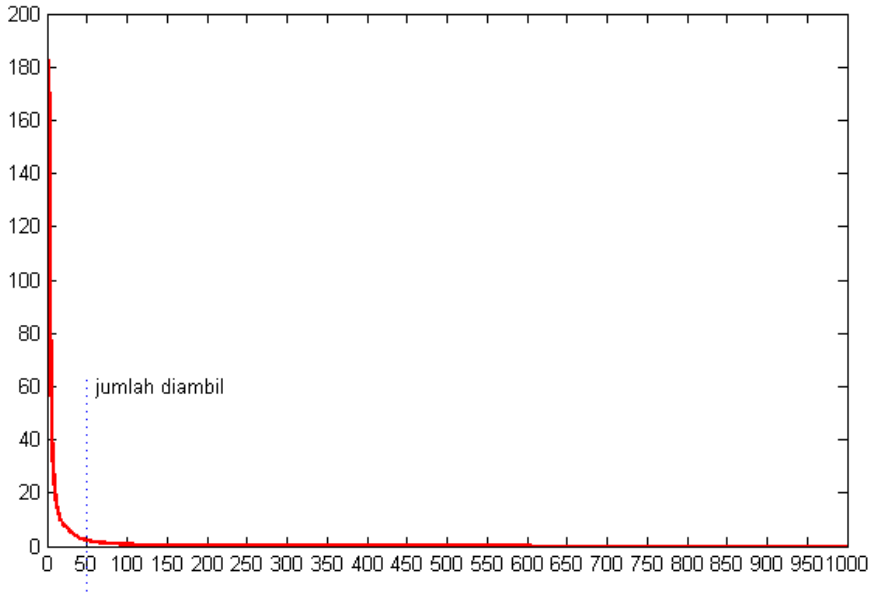
$$\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_{n^2} \end{bmatrix} \rightarrow \begin{bmatrix} x_{1,1} & x_{2,1} & \dots & x_{n^2,1} \\ x_{1,2} & x_{2,2} & \dots & x_{n^2,2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1,n^2} & x_{2,n^2} & \dots & x_{n^2,n^2} \end{bmatrix} = \begin{bmatrix} \lambda_1 & x_{1,1} & x_{2,1} & \dots & x_{n^2,1} \\ \lambda_2 & x_{1,2} & x_{2,2} & \dots & x_{n^2,2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \lambda_{n^2} & x_{1,n^2} & x_{2,n^2} & \dots & x_{n^2,n^2} \end{bmatrix} \downarrow urutkan \quad (2\text{-}54)$$

Setelah mendapatkan matriks dengan vektor Eigen terurut berdasarkan nilai Eigen terbesar, buang kolom pertama yang merupakan informasi nilai Eigen.

$$\begin{bmatrix} \lambda_{s1} \\ \lambda_{s2} \\ \vdots \\ \lambda_{sn^2} \end{bmatrix} \leftarrow \begin{bmatrix} s_{1,1} & s_{1,2} & \dots & s_{1,n^2} \\ s_{2,1} & s_{2,2} & \dots & s_{2,n^2} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n^2,1} & s_{n^2,2} & \dots & s_{n^2,n^2} \end{bmatrix} \quad (2\text{-}55)$$

6. Potong dimensi

Tentukan berapa jumlah dimensi (p) yang akan diambil. Sebagai bahan acuan untuk mengambil berapa banyak dimensi yang diambil dapat dilakukan *plotting* nilai Eigen ke dalam sebuah *chart.*Berikut adalah contoh data yang dipakai pada penelitian ini, citra wajah dengan ukuran dimensi 32 x 32 piksel.



Gambar 2. 12 Urutan Nilai Eigen dari Data

Dengan melihat plot diatas, maka ditentukan jumlah dimensi yang diambil adalah p = 50. Setelah didapatkan jumlah dimensi yang diambil, maka ambil p baris teratas pada matriks yang telah terurut berdasarkan nilai Eigen terbesar dan lakukan *transpose* pada matriks tersebut. Terbentuklah matriks transformasi yang berukuran p x n^2

$$TransMat = \begin{bmatrix} s_{1,1} & s_{1,2} & \cdots & s_{1,p} \\ s_{2,1} & s_{2,2} & \cdots & s_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n^2,1} & s_{n^2,2} & \cdots & s_{n^2,p} \end{bmatrix} \quad (2\text{-}56)$$

7. Kalikan data acuan yang ternormalisasi dengan matriks transformasi

Langkah terakhir untuk mereduksi data acuan adalah dengan mengalikan data acuan yang telah ternormalisasi (k x n^2) dengan matriks transformasi (n^2xp) akan menghasilkan data acuan baru dengan jumlah dimensi yang lebih kecil (k x p), dimana p < n^2.

$$ImgMatNorm\ x\ TransMat = RedDataTrain \quad (2\text{-}57)$$

$$\begin{bmatrix} z_{1,1} & z_{1,2} & \dots & z_{1,n^2} \\ z_{2,1} & z_{2,2} & \dots & z_{2,n^2} \\ \vdots & \vdots & \ddots & \vdots \\ z_{k,1} & z_{k,2} & \dots & z_{k,n^2} \end{bmatrix} \begin{bmatrix} s_{1,1} & s_{1,2} & \dots & s_{1,p} \\ s_{2,1} & s_{2,2} & \dots & s_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n^2,1} & s_{n^2,2} & \dots & s_{n^2,p} \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,p} \\ a_{2,1} & a_{2,2} & \dots & a_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k,1} & a_{k,2} & \dots & a_{k,p} \end{bmatrix}$$

(2-58)

2.2.3. Reduksi Data Uji

Berikut ini adalah langkah untuk mereduksi data uji:

1. Susun data uji

Data uji dibuat sebagai vektor T berukuran 1 x n^2

$$T = (t_1, t_2, \dots t_{n^2}) \quad (2\text{-}59)$$

2. Lakukan normalisasi dengan zscore

Dengan menggunakan rata-rata dan standar deviasi yang didapatkan dari data acuan, lakukan normalisasi pada vektor data uji.

$$meanVec = (\bar{x}_1, \bar{x}_2, \dots, \bar{x}_{n^2}) \quad (2\text{-}60)$$

$$stdVec = (\sigma_1, \sigma_2, \dots, \sigma_{n^2}) \quad (2\text{-}61)$$

$$zt_i = \frac{t_i - \bar{x}_i}{\sigma_i} \quad (2\text{-}62)$$

$$DataTestNorm = (zt_1, zt_2, \dots, zt_{n^2}) \quad (2\text{-}63)$$

3. Kalikan data uji yang ternormalisasi dengan matriks transformasi

Langkah terakhir untuk mereduksi data uji adalah dengan mengalikan data uji yang telah ternormalisasi (1 x n^2) dengan matriks transformasi (n^2xp) akan menghasilkan data uji baru dengan jumlah dimensi yang lebih kecil (1 x p), dimana $p < n^2$.

$$DataTestNorm\, x\, TransMat = RedDataTest \quad (2\text{-}64)$$

$$[zt_{1,1} \quad zt_{1,2} \quad \dots \quad zt_{1,n^2}] \begin{bmatrix} s_{1,1} & s_{1,2} & \dots & s_{1,p} \\ s_{2,1} & s_{2,2} & \dots & s_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n^2,1} & s_{n^2,2} & \dots & s_{n^2,p} \end{bmatrix} = [b_{1,1} \quad b_{1,2} \quad \dots \quad b_{1,p}]$$

(2-65)

Semua langkah-langkah yang telah dijelaskan diatas diimplementasikan dengan menggunakan MATLAB untuk digunakan pada penelitian ini.

2.3. *Noise*

Dalam penelitian-penelitian yang sudah dilakukan sebelumnya, citra wajah yang digunakan baik sebagai data acuan dan data uji merupakan citra gambar yang ideal atau normal. Padahal dalam kenyataannya, citra wajah yang didapat keadaannya tidak selalu ideal, sehingga belum dapat mewakili keadaan yang sebenarnya.

Untuk itu dalam penelitian kali ini, penulis melaksanakan percobaan dengan citra wajah yang terdegradasi oleh *noise*. Skema pemberian *noise* akan dijelaskan lebih rinci pada Bab 3. Jenis *noise* yang digunakan ada empat, yaitu: Gaussian, Poisson, Salt & Pepper, serta Speckle.

2.3.1. Gaussian

Noise Gaussian adalah jenis *white noise* dengan distribusi yang normal. *White noise* adalah fluktuasi secara acak dari sinyal (dalah hal ini *grayscale*). Untuk menambahkan *noise* ini ke citra cukup dengan menambahkan matriks citra wajah dengan matriks yang berisi angka *random* yang terdistribusi normal. Dalam eksperimen ini, digunakan rata-rata dan variansi standar dari fungsi imnoise(data,'gaussian') MATLAB, yaitu nilai rata-rata 0 dan variansi 0.01.

2.3.2. Poisson

Noise ini dibuat berdasarkan dari distribusi poisson. Pemberian noise dilakukan dengan fungsi imnoise(data,'poisson') pada MATLAB.

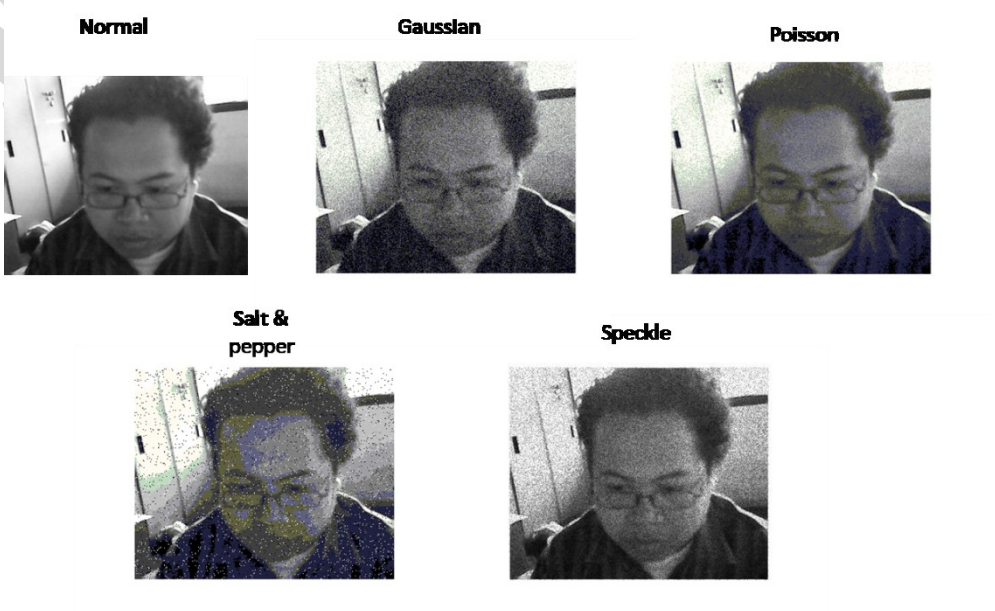
2.3.3. Salt & Pepper

Noise jenis ini dapat dianggap sebagai gangguan acak yang ekstrim pada data, yaitu titik-titik hitam atau putih yang tidak berhubungan dengan keadaan sinyal di sekitarnya. Dalam eksperimen ini, digunakan intensitas standar dari fungsi imnoise(data,'salt & pepper') MATLAB, yaitu nilai intensitas 0.1 atau 10 persen dari keseluruhan data.

2.3.4. Speckle

Noise speckle sedikit banyak memiliki persamaan dengan *noise* Gaussian, yaitu berupa angka acak yang terdistribusi normal. Perbedaan dengan *noise* Gaussian adalah untuk mengenai citra wajah, angka-angka acak tersebut harus dikalikan dengan matriks wajah. Dalam eksperimen ini, digunakan rata-rata dan variansi standar dari fungsi imnoise(data,'speckle') MATLAB, yaitu nilai rata-rata 0 dan variansi 0.04.

Berikut ini masing-masing contoh gambar yang belum dan sudah dikenai *noise*:



Gambar 2. 13 Contoh Hasil Citra yang Terdegradasi *Noise*

Pemberian *noise* pada percobaan ini bervariasi, dari pemberian secara seragam maupun acak.