BAB IV

HASIL DAN PEMBAHASAN

4.1. Deskripsi Data

Data yang digunakan dalam penelitian ini adalah dataset STARE. Berjumlah 20 citra retina yang diperoleh dari data penelitian yang dilakukan oleh Shiley Eye Center dari University Of California dapat diakses melalui laman https://cecas.clemson.edu/~ahoover/stare/. Adapun contoh beberapa dataset tersebut dijelaskan dalam Tabel 4.1.

Tabel 4. 1 Beberapa contoh dataset STARE

No	Nama file	Citra	Label
1.	Im0198		Normal
2.	Im0186		*Diabetic Retinopathy*
3.	Im0340		*Diabetic Retinopathy*
4.	Im0347		*Diabetic Retinopathy*
5.	Im0349		*Diabetic Retinopathy*

Dataset STARE berukuran dimensi 605×700 piksel yang berisi 10 citra normal dan 10 citra DR dengan berformat ppm.

4.2. *Pre-processing*

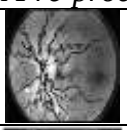
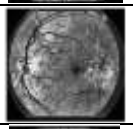
Proses *pre-processing* dilakukan dengan tujuan agar data yang digunakan dapat optimal dalam proses pengklasifikasian. *Input* data yang digunakan dalam

penelitian ini berupa citra RGB dari dataset STARE dengan ukuran dimensi yaitu 605×700 piksel. Adapun tahapan *pre-processing* sebagai berikut :

1. Proses pertama dalam *pre-processing* adalah *input* data berupa citra RGB.
2. Gunakan metode *cropping image* untuk memotong bagian background gambar, selanjutnya *resize* data dengan ukuran 336×336.
3. Setelah itu untuk memperoleh kontras terbaik pada citra sehingga digunakan *green channel*.
4. Setelah itu dilakukan proses *CLAHE* yang bertujuan untuk membuat jaringan pembuluh darah retina terlihat lebih jelas.

Beberapa contoh hasil citra yang telah di *pre-processing* dapat dilihat pada Tabel 4.2.

Tabel 4. 2 Beberapa *contoh hasil dataset yang telah di* pre-processing

No	Nama file	Citra Asli	Setelah *Pre-processing*
1.	Im0198		
2.	Im0186		
3.	Im0340		
4.	Im0347		
5.	Im0349		

Pada tabel terlihat perbedaan citra sebelum dan sesudah di *pre-processing*, pembuluh darah pada retina semakin telihat lebih jelas. Hasil citra yang telah di *pre-processing* akan dilanjutkan ke tahap selanjutnya yaitu augmentasi.

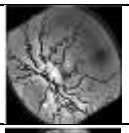
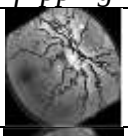
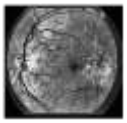
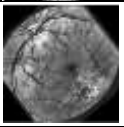
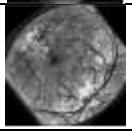
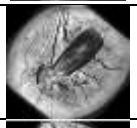
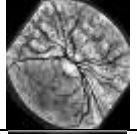
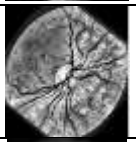
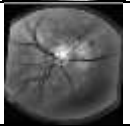
4.3. Augmentasi

Proses augmentasi bertujuan untuk mengatasi permasalahan keterbatasan data pada saat penelitian dan membantu meningkatkan kinerja model. Adapun tahapan augmentasi sebagai berikut :

1. Proses awal pada augmentasi yaitu input citra yang telah di *pre-processing* dengan ukuran 336×336.
2. Tiap citra yang diinput akan dilakukan metode *rotation image* dengan sudut acak antara $1°$ sampai dengan $179°$.
3. Hasil tiap citra yang telah dirotasi akan dilakukan metode *flipping image* secara vertikal ataupun horizontal.

Beberapa contoh gambar yang telah dilakukan *rotation image* dan *flipping image* secara dapat dilihat pada Tabel 4.3.

Tabel 4. 3 Beberapa contoh citra yang telah dilakukan augmentasi

No.	Nama File	Citra pre-processing	Citra hasil rotasi	Citra hasil *flipping*
1.	Im0198			
2.	Im0186			
3.	Im0340			
4.	Im0347			
5.	Im0349			

Pada Tabel 4.3 dapat dilihat tidak ada perubahan ciri dari citra tersebut. Dari 1 citra asli dapat menjadi beberapa citra baru setelah dilakukan *rotation image* dengan sudut acak. Citra yang telah dilakukan *rotation image* selanjutnya akan dilakukan metode *flipping image* secara vertikal ataupun horizontal. Pada proses augmentasi menghasilkan data baru berjumlah 18000 data dimana data tersebut akan dibagi menjadi dua kategori yaitu data *training* dan data *testing*.

4.4. Klasifikasi Citra

Dalam arsitektur CNN terdapat beberapa proses yang biasanya dilakukan seperti *padding same*, operasi *convolution*, *convolution, depthwise convolution*, *pointwise convolution*, fungsi aktivasi, *batch normalization*, *max pooling*, *concatenate.* Adapun contoh proses perhitungan manual tersebut sebagai berikut :

1. *Padding same*

Padding same merupakan parameter dalam *convolution layer* yang digunakan untuk membuat ukuran matriks *input* dan martiks *output* berukuran sama. Pada *padding same* dilakukan penambahan entri 0 pada sekeliling entri matriks *input*. Berikut contoh proses *padding same* dapat dilihat sebagai berikut :

Contoh *padding same* :

Sebagai contoh diambil matriks *input* A berukuran 3×3 yaitu :

$$A = \begin{bmatrix} 1 & 3 & 4 \\ 8 & 6 & 12 \\ 5 & 6 & 3 \end{bmatrix}$$

Kemudian lakukan *padding same* pada matriks *input* A dengan menambahkan nilai entri 0 pada sekeliling entri matriks A sehingga menjadi :

$$A = \begin{bmatrix} 1 & 3 & 4 \\ 8 & 6 & 12 \\ 5 & 6 & 3 \end{bmatrix} \Rightarrow A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 3 & 4 & 0 \\ 0 & 8 & 6 & 12 & 0 \\ 0 & 5 & 6 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Pada contoh *padding same* yang awalnya matriks A berukuran 3×3 menjadi 5×5.

2. Operasi *convolution*

Berdasarkan Gambar 2.1. dalam proses operasi *convolution* terdapat beberapa parameter yang digunakan yaitu jumlah filter, *stride*, *padding*, dan matriks kernel berukuran $n \times n$. Berikut contoh operasi *convolution* :

Contoh operasi *convolution* :

Misalkan diambil matriks A yang telah dilakukan *padding same* pada contoh *padding same* sebagai *input* ke-1 (A_1). Kemudian misalkan *stride* 1, *padding same*, matriks kernel ke-1 (K_1) berukuran 3×3 yaitu :

$$K_1 = \begin{bmatrix} -1 & 0 & 1 \\ 0 & 1 & -1 \\ -1 & 1 & 0 \end{bmatrix}$$

Lakukan perhitungan operasi *convolution* A_1^1 menggunakan Persamaan (2.1) sebagai berikut :

$$A_1^1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 3 & 4 & 0 \\ 0 & 8 & 6 & 12 & 0 \\ 0 & 5 & 6 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \\ 0 & 1 & -1 \\ -1 & 1 & 0 \end{bmatrix}$$

Dimana untuk perhitungan setiap entri matriks operasi *convolution* A_1^1 menggunakan Persamaan (2.2).

$$a_{i,j} = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} \left(c_{u+i,v+j} \times k_{u+i,v+j}\right)$$

Untuk $n = 3$, $i = 1,2,3$, dan $j = 1,2,3$

a. Hitung hasil *convolution* untuk entri matriks A_1^1 baris ke-1 kolom ke-1.

$$a_{1,1} = \sum_{u=0}^{3-1} \sum_{v=0}^{3-1} \left(c_{u+1,v+1} \times k_{u+1,v+1}\right)$$

$$= \sum_{u=0}^{2} \sum_{v=0}^{2} \left(c_{u+1,v+1} \times k_{u+1,v+1}\right)$$

$$= \left(c_{0+1,0+1} \times k_{0+1,0+1}\right) + \left(c_{0+1,1+1} \times k_{0+1,1+1}\right) + \left(c_{0+1,2+1} \times k_{0+1,2+1}\right)$$
$$+ \left(c_{1+1,0+1} \times k_{1+1,0+1}\right) + \left(c_{1+1,1+1} \times k_{1+1,1+1}\right)$$
$$+ \left(c_{1+1,2+1} \times k_{1+1,2+1}\right) + \left(c_{2+1,0+1} \times k_{2+1,0+1}\right)$$
$$+ \left(c_{2+1,1+1} \times k_{2+1,1+1}\right) + \left(c_{2+1,2+1} \times k_{2+1,2+1}\right)$$

$$= \left(c_{1,1} \times k_{1,1}\right) + \left(c_{1,2} \times k_{1,2}\right) + \left(c_{1,3} \times k_{1,3}\right) + \left(c_{2,1} \times k_{2,1}\right)$$
$$+ \left(c_{2,2} \times k_{2,2}\right) + \left(c_{2,3} \times k_{2,3}\right) + \left(c_{3,1} \times k_{3,1}\right)$$
$$+ \left(c_{3,2} \times k_{3,2}\right) + \left(c_{3,3} \times k_{3,3}\right)$$

$$= \left(0 \times (-1)\right) + (0 \times 0) + (0 \times 1) + (0 \times 0) + (1 \times 1) + (3 \times (-1))$$
$$+ \left(0 \times (-1)\right) + (8 \times 1) + (6 \times 0)$$

$$= 8 + 1 - 3$$

$$= 6$$

b. Lakukan proses perhitungan pada setiap entri baris dan kolom selanjutnya sampai baris ke-3 dan kolom ke-3. Sehingga diperoleh A_1^1 matriks sebagai berikut :

$$A_1^1 = \begin{bmatrix} 6 & -3 & 10 \\ 10 & -2 & -6 \\ 5 & 7 & -3 \end{bmatrix}$$

3. *Convolution*

Untuk menghitung hasil *convolution* dapat dilihat pada Contoh *Convolution* dengan menggunakan Persamaan (2.3).

Contoh *Convolution* :

Misalkan diambil matriks *A* yang telah dilakukan *padding same* pada Contoh *Padding same* sebagai *input* ke-1 (A_1) dan matriks kernel ke-1 (K_1) dari Contoh Operasi *Convolution*, dimana $i = 1,2,3$ dan $j = 1,2,3$. Tentukan nilai bias kernel ke-1 $b_1 = 0.1$. Kemudian lakukan perhitungan untuk setiap entri matriks *convolution input* ke-1 pada kernel ke-1 C_1^1 menggunakan Persamaan (2.3) sebagai berikut:

$$a_{i,j} = \left(\sum_{u=0}^{n-1} \sum_{v=0}^{n-1} \left(c_{u+i,v+j} \times k_{u+i,v+j} \right) \right) + b_q$$

Untuk $n = 3$, $i = 1,2,3$, dan $j = 1,2,3$

a. Hitung hasil *convolution* untuk entri matriks A_1^1 baris ke-1 kolom ke-1.

$$a_{1,1} = \left(\sum_{u=0}^{3-1} \sum_{v=0}^{3-1} \left(c_{u+1,v+1} \times k_{u+1,v+1} \right) \right) + b_q$$

$$= \left(\sum_{u=0}^{2} \sum_{v=0}^{2} \left(c_{u+1,v+1} \times k_{u+1,v+1} \right) \right) + b_q$$

$$= \Big(\big(c_{0+1,0+1} \times k_{0+1,0+1} \big) + \big(c_{0+1,1+1} \times k_{0+1,1+1} \big) + \big(c_{0+1,2+1} \times k_{0+1,2+1} \big) + \big(c_{1+1,0+1} \times k_{1+1,0+1} \big) + \big(c_{1+1,1+1} \times k_{1+1,1+1} \big) + \big(c_{1+1,2+1} \times k_{1+1,2+1} \big) + \big(c_{2+1,0+1} \times k_{2+1,0+1} \big) + \big(c_{2+1,1+1} \times k_{2+1,1+1} \big) + \big(c_{2+1,2+1} \times k_{2+1,2+1} \big) \Big) + b_q$$

$$= \big(\big(c_{1,1} \times k_{1,1} \big) + \big(c_{1,2} \times k_{1,2} \big) + \big(c_{1,3} \times k_{1,3} \big) + \big(c_{2,1} \times k_{2,1} \big) + \big(c_{2,2} \times k_{2,2} \big) + \big(c_{2,3} \times k_{2,3} \big) + \big(c_{3,1} \times k_{3,1} \big) + \big(c_{3,2} \times k_{3,2} \big) + \big(c_{3,3} \times k_{3,3} \big) \big) + b_q$$

$$= \Big(\big(0 \times (-1) \big) + (0 \times 0) + (0 \times 1) + (0 \times 0) + (1 \times 1) + \big(3 \times (-1) \big) + \big(0 \times (-1) \big) + (8 \times 1) + (6 \times 0) \Big) + 0.1$$

$$= (8 + 1 - 3) + 0.1$$

$$= 6 + 0.1$$

$$= 6.1$$

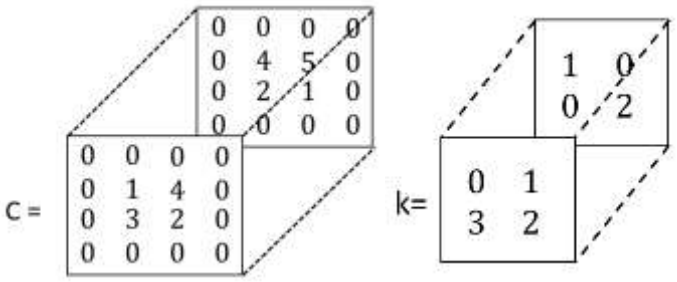
b. Lakukan proses perhitungan pada setiap entri baris dan kolom selanjutnya sampai baris ke-3 dan kolom ke-3. Sehingga diperoleh matriks B_1^1 sebagai berikut :

$$B_1^1 = \begin{bmatrix} 6{,}1 & -2{,}9 & 10{,}1 \\ 10{,}1 & -1{,}9 & -5{,}9 \\ 5{,}1 & 7{,}1 & -2{,}9 \end{bmatrix}$$

4. *Depthwise Seperable Convolution*

a. *Depthwise Convolution*

Misalkan diambil matriks 3 dimensi C berukuran 4x4x2 yang telah dilakukan *padding same*, dan terdapat kernel 2x2 dimana i=1,2,3 dan j=1,2,3. Matriks C dan kernel dapat dilihat sebagai berikut:



Matriks C akan dipisah menjadi 2 matriks yang berukuran 4×4 dan kernel dipisah menjadi 2×2. Sehingga diperoleh matriks sebagai berikut :

$$C_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 0 \\ 0 & 3 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad C_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 4 & 5 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$k_1 = \begin{bmatrix} 0 & 1 \\ 3 & 2 \end{bmatrix} \qquad k_2 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

Selanjutnya akan dilakukan proses *convolution* seperti pada pada Persamaan (2.11), misalkan diambil matriks C_1 sebagai matriks input ke-1, *stride* sebanyak 1, dan matriks kernel ke-1 sebagai berikut :

$$D_{i_{(i,j)}} = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} \left(\left(a_{k_{(u+i,v+j)}} \right) \left(b_{k_{(u+1,v+1)}} \right) \right)$$

$$= \sum_{u=0}^{3-1} \sum_{v=0}^{3-1} \left(\left(a_{1_{(u+i,v+j)}} \right) \left(b_{1_{(u+1,v+1)}} \right) \right)$$

$$= \left(a_{1_{(1,1)}} \times b_{1_{(1,1)}}\right) + \left(a_{1_{(1,2)}} \times b_{1_{(1,2)}}\right) + \left(a_{1_{(2,1)}} \times b_{1_{(2,1)}}\right) + \left(a_{1_{(2,2)}} \times b_{1_{(2,2)}}\right)$$

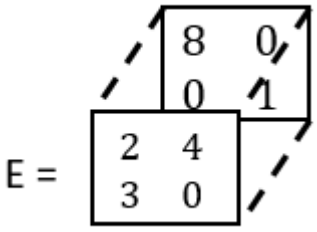
$$= (0 * 0) + (0 * 1) + (0 * 3) + (0 * 2)$$

$$= 2$$

Lakukan proses yang sama pada entri baris dan kolom selanjutnya serta pada matriks C_2. Maka akan diperoleh hasil perhitungan *depthwise convolution* adalah sebagai berikut:

$$G_1 = \begin{bmatrix} 2 & 4 \\ 3 & 0 \end{bmatrix} \qquad G_2 = \begin{bmatrix} 8 & 0 \\ 0 & 1 \end{bmatrix}$$

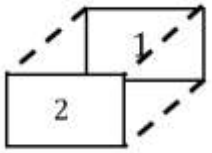
Kemudian dilakukan proses *concantened* pada kedua matriks sesuai perhitungan pada Subbab 4.4.10 menghasilkan matriks 3 dimensi baru.

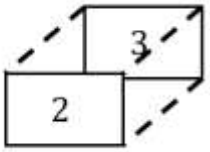


b. *Pointwise convolution*

Misalkan diambil matriks E berukuran $2 \times 2 \times 2$ yang diambil dari proses *depthwise convolution*. Misalkan diambil kernel *k* yang terdiri dari kernel t dan s berukuran $1 \times 1 \times 2$ dengan baris $i = 1,2$. dan kolom $j = 1,2$. Matriks E akan dipisah menjadi 2 matriks berukuran $2 \times 2 \times 1$ yang terdiri dari matriks E_1 dan E_2.

$$E_1 = \begin{bmatrix} 2 & 4 \\ 3 & 0 \end{bmatrix} \qquad E_2 = \begin{bmatrix} 8 & 0 \\ 0 & 1 \end{bmatrix}$$

t = 

s = 

Lakukan proses konvolusi seperti pada Persamaan (2.12), Ambil matriks E_1 sebagai matriks *input* ke-1 dan matriks kernel t berukuran 1×1 sebagai berikut:

$$Pc = \sum_{u=0}^{n-1} (W_{i_{(i,j,u+1)}} \times x_{i_{(1,j,u+1)}})$$

$$= \sum_{u=0}^{2-1} (W_{i_{(i,j,u+1)}} \times x_{i_{(1,j,u+1)}})$$

$$= \sum_{u=0}^{1} (W_{i_{(i,j,u+1)}} \times x_{i_{(1,j,u+1)}})$$

$$= (W_{1_{(1,1,0+1)}} \times x_{1_{(1,1,0+1)}}) + (W_{2_{(1,1,1+1)}} \times x_{2_{(1,1,1+1)}})$$

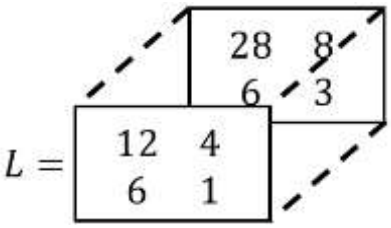
$$= (W_{1_{(1,1,1)}} \times x_{1_{(1,1,1)}}) + (W_{2_{(1,1,2)}} \times x_{2_{(1,1,2)}})$$

$$= (2 \times 2) + (1 \times 8)$$

$$= 4 + 8$$

$$= 12$$

Lakukan hal yang sama pada entri matriks berikutnya dan lakukan juga pada kernel s. Sehingga diperoleh matriks L sebagai berikut:



5. *Batch Normalization*

Untuk melakukan perhitungan *batch normalization* dapat dilihat pada Contoh berikut dengan menggunakan Persamaan (2.7), (2,8), (2,9).

Contoh batch normalization :

Untuk menghitung *batch normalization* misalkan *input* yang akan digunakan diambil dari matriks *X* pada Contoh Proses *ReLU* yaitu:

$$X = \begin{bmatrix} 6.1 & 0 & 10.1 \\ 10.1 & 0 & 0 \\ 5.1 & 7.1 & 0 \end{bmatrix}$$

Matriks X untuk setiap entrinya akan dilakukan normalisasi menggunakan *batch normlization* dengan langkah-langkah sebagai berikut:

a. Tentukan jumlah *mini batch* (n) dan jumlah data dalam *mini batch* (m) dari matriks *input*. Berdasarkan matriks X yang berukuran 3×3 diperoleh jumlah data dalam *mini batch* $(m) = 3$ dan jumlah *mini batch* $(n) = 3$ dengan $i = 1,2,3$ dan $j = 1,2,3$.

b. Hitung rata-rata untuk setiap *mini batch* (μ_j) menggunakan Persamaan (2.7). Contoh perhitungan secara manual dapat dilihat sebagai berikut :

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\mu_1 = \frac{1}{3}\sum_{i=1}^{3} x_i = \frac{1}{3}(6,1 + 10,1 + 5,1) = \frac{1}{3}(21,3) = 7.1$$

$$\mu_2 = \frac{1}{3}\sum_{i=1}^{3} x_i \ = \frac{1}{3}(0 + 0 + 7{,}1) = \frac{1}{3}(7{,}1) = 2.36$$

$$\mu_3 = \frac{1}{3}\sum_{i=1}^{3} x_i \ = \frac{1}{3}(10{,}1 + 0 + 0) = \frac{1}{3}(10{,}1) = 3.36$$

c. Hitung variansi untuk setiap *mini batch* (σ_j^2) menggunakan Persamaan (2.8), Contoh perhitungan secara manual dapat dilihat dibawah ini:

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_j)^2$$

$$\sigma_1^2 = \frac{1}{3}\sum_{i=1}^{3}\left(x_i - \mu_j\right)^2$$

$$= \frac{1}{3}[(6.1 - 7.1)^2 + (10.1 - 7.1)^2 + (5.1 - 7.1)^2]$$

$$= \frac{1}{3}[(-1)^2 + (3)^2 + (-2)^2]$$

$$= \frac{1}{3}[1 + 9 + 4]$$

$$= \frac{1}{3}[14]$$

$$= 4.67$$

$$\sigma_2^2 = \frac{1}{3}\sum_{i=1}^{3}\left(x_i - \mu_j\right)^2$$

$$= \frac{1}{3}[(0 - 2.36)^2 + (0 - 2.36)^2 + (7.1 - 2.36)^2]$$

$$= \frac{1}{3}[(-2.36)^2 + (-2.36)^2 + (4.74)^2]$$

$$= \frac{1}{3}[5.56 + 5.56 + 22.46]$$

$$= \frac{1}{3}[33.58]$$

$$= 11.19$$

$$\sigma_3^2 = \frac{1}{3}\sum_{i=1}^{3}(x_i - \mu_j)^2$$

$$= \frac{1}{3}[(10.1 - 3.36)^2 + (0 - 3.36)^2 + (0 - 3.36)^2]$$

$$= \frac{1}{3}[(6{,}74)^2 + (-3{,}36)^2 + (-3{,}36)^2]$$

$$= \frac{1}{3}[45{,}42 + 11{,}29 + 11{,}29]$$

$$= \frac{1}{3}[68]$$

$$= 22{,}67$$

d. Hitung normalisasi untuk setiap entri matriks $\hat{X}$ menggunakan Persamaan (2.9) dengan memisalkan nilai $\varepsilon = 10^{-5}$. Perhitungan secara manual untuk tiap entri matriks $\hat{X}$ dapat dilihat dibawah ini:

$$\widehat{x_{i,j}} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

$$\widehat{x_{1,1}} = \frac{x_{1,1} - \mu_1}{\sqrt{\sigma_1^2 + \varepsilon}} = \frac{6{,}1 - 7{,}1}{\sqrt{4{,}67 + 10^{-5}}} = \frac{-1}{2{,}16} = -0{,}46$$

$$\widehat{x_{1,2}} = \frac{x_{1,2} - \mu_2}{\sqrt{\sigma_2^2 + \varepsilon}} = \frac{0 - 2{,}36}{\sqrt{11{,}19 + 10^{-5}}} = \frac{-2{,}36}{3{,}34} = -0{,}71$$

$$\widehat{x_{1,3}} = \frac{x_{1,3} - \mu_3}{\sqrt{\sigma_3^2 + \varepsilon}} = \frac{10{,}1 - 3{,}36}{\sqrt{22{,}67 + 10^{-5}}} = \frac{6{,}74}{4{,}76} = 1{,}42$$

$$\widehat{x_{2,1}} = \frac{x_{2,1} - \mu_1}{\sqrt{\sigma_1^2 + \varepsilon}} = \frac{10{,}1 - 7{,}1}{\sqrt{4{,}67 + 10^{-5}}} = \frac{3}{2{,}16} = 1{,}38$$

$$\widehat{x_{2,2}} = \frac{x_{2,2} - \mu_2}{\sqrt{\sigma_2^2 + \varepsilon}} = \frac{0 - 2{,}36}{\sqrt{11{,}19 + 10^{-5}}} = \frac{-2{,}36}{3{,}34} = -0{,}71$$

$$\widehat{x_{2,3}} = \frac{x_{2,3} - \mu_3}{\sqrt{\sigma_3^2 + \varepsilon}} = \frac{0 - 3{,}36}{\sqrt{22{,}67 + 10^{-5}}} = \frac{-3{,}36}{4{,}76} = -0{,}70$$

$$\widehat{x_{3,1}} = \frac{x_{3,1} - \mu_1}{\sqrt{\sigma_1^2 + \varepsilon}} = \frac{5{,}1 - 7{,}1}{\sqrt{4{,}67 + 10^{-5}}} = \frac{-2}{2{,}16} = -0{,}92$$

$$\widehat{x_{3,2}} = \frac{x_{3,2} - \mu_2}{\sqrt{\sigma_2^2 + \varepsilon}} = \frac{7{,}1 - 2{,}36}{\sqrt{11{,}19 + 10^{-5}}} = \frac{4{,}74}{3{,}34} = 1{,}42$$

$$\widehat{x_{3,3}} = \frac{x_{3,3} - \mu_3}{\sqrt{\sigma_3^2 + \varepsilon}} = \frac{0 - 3{,}36}{\sqrt{22{,}67 + 10^{-5}}} = \frac{-3{,}36}{4{,}76} = -0{,}70$$

Jadi hasil perhitungan *batch normalization* secara keseluruhan dapat dilihat pada matriks $\hat{X}$ berikut:

$$\hat{X} = \begin{bmatrix} \widehat{X_{1,1}} & \widehat{X_{1,2}} & \widehat{X_{1,3}} \\ \widehat{X_{2,1}} & \widehat{X_{2,2}} & \widehat{X_{2,3}} \\ \widehat{X_{3,1}} & \widehat{X_{3,2}} & \widehat{X_{3,3}} \end{bmatrix} = \begin{bmatrix} -0{,}46 & -0{,}71 & 1{,}42 \\ 1{,}38 & -0{,}71 & -0{,}70 \\ -0{,}92 & 1{,}42 & -0{,}70 \end{bmatrix}$$

6. ReLU

Pada tahap ini dilakukan subsitusi matriks *input* ke dalam fungsi aktivasi *ReLU* menggunakan Persamaan (2.4).

Contoh ReLU :

Misalkan diambil matriks A_1^1 dari hasil Contoh *Convolution*, maka entri

matriks $c_{1,1} = 6{,}1$. Kemudian entri matriks$c_{1,1}$ disubtusikan kedalam fungsi aktivasi *ReLU* $R(z)$ sehingga diperoleh:

$$R(6{,}1) = \max\big(0, (6{,}1)\big) = 6{,}1$$

Untuk entri matriks A_1^1 lainnya yang telah disubsitusikan ke dalam fungsi aktivasi *ReLU* dapat dilihat pada matriks *X* berikut :

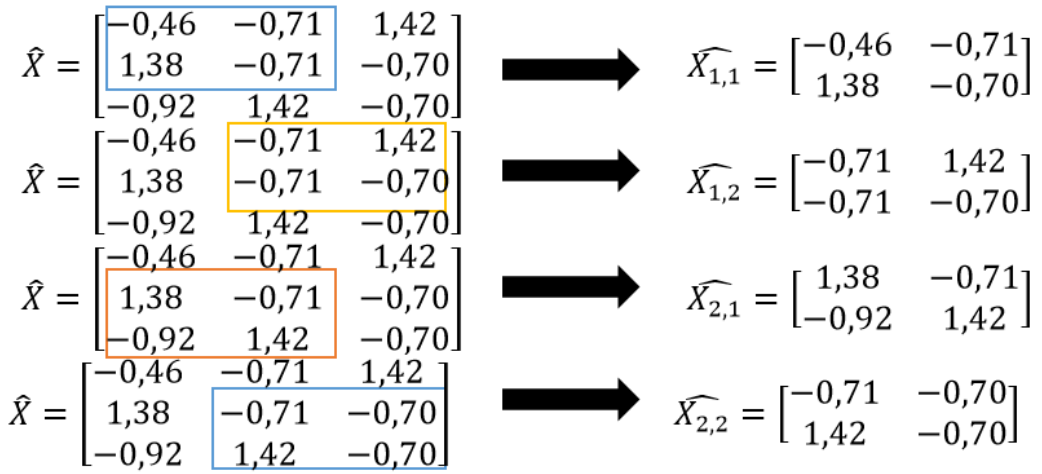
$$X = \begin{bmatrix} 6{,}1 & 0 & 10{,}1 \\ 10{,}1 & 0 & 0 \\ 5{,}1 & 7{,}1 & 0 \end{bmatrix}$$

7. *Max pooling*

Berdasarkan Gambar 2.7, proses *max pooling* digunakan untuk mengurangi ukuran pada matriks *input*. Cara kerja *max pooling* yaitu dengan mempartisi matriks *input* menjadi sub-sub matriks berukuran $n \times n$ sesuai dengan ukuran filter *max pooling* yang digunakan dan pergesaran *stride* secara horizontal maupun vertikal.

Contoh *max pooling* :

Misalkan diambil matriks *X* pada Contoh Proses *Batch Normalization* sebagai matriks *input*. Kemudian tentukan ukuran filter *max pooling* dan *stride* yang digunakan. Sebagai contoh diambil filter *max pooling* berukuran 2×2 dan *stride* 1, sehingga matriks *input X* akan dipartisi menjadi sub-sub matriks yang berukuran 2×2 dengan pergesaran *stride* secara horizontal maupun vertikal sebesar 1. Adapun proses partisi sub-sub matriks dapat dilihat sebagai berikut ini:



Dari contoh diatas terlihat kotak berwarna biru yang menunjukkan matriks filter *max pooling* berukuran 2×2 untuk mendapatkan submatriks $\hat{X}_{1,1}$. Geser filter *max pooling* 1 langkah ke kanan (horizontal) untuk mendapatkan submatriks $\hat{X}_{1,2}$ yang ditunjukkan oleh kotak berwarna oranye. Geser filter *max pooling* sebesar 1 langkah ke bawah (vertikal) dan ke kanan (horizontal) untuk mendapatkan submatriks $\hat{X}_{2,1}$, dan $\hat{X}_{2,2}$, ditunjukkan oleh kotak berwarna merah dan biru. Dari proses tersebut diperoleh 4 submatriks yaitu menunjukkan matriks filter *max pooling* berukuran 2×2 untuk mendapatkan submatriks $\hat{X}_{1,1}$, $\hat{X}_{1,2}$, $\hat{X}_{2,1}$, dan $\hat{X}_{2,2}$. Selanjutnya tentukan nilai maksimumnya untuk masing-masing submatriks sehingga diperoleh:

$$m_{1,1} = maks\left(\hat{X}_{1,1},\right) = 1{,}38$$

$$m_{1,2} = maks(\hat{X}_{1,2)} = 1{,}42$$

$$m_{2,1} = maks(\hat{X}_{2,1}) = 1{,}38$$

$$m_{2,2} = maks(\hat{X}_{2,2)} = 1{,}42$$

Dari keempat nilai maksimum pada masing-masing submatriks akan dimasukkan kedalam sebuah matriks M berukuran 2×2 yang merupakan hasil dari

operasi *max pooling*. Matriks M tersebut dapat dilihat sebagai berikut:

$$M = \begin{bmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{bmatrix} = \begin{bmatrix} 1{,}38 & 1{,}42 \\ 1{,}38 & 1{,}42 \end{bmatrix}$$

8. Avarage Pooling

Pada tahap ini akan dilakukan digunakan untuk mengurangi ukuran pada matriks *input*. Cara kerja *avarage pooling* yaitu dengan mempartisi matriks *input* menjadi sub-sub matriks berukuran $n \times n$ sesuai dengan ukuran filter *avarage pooling* yang digunakan dan pergesaran *stride* secara horizontal maupun vertikal.

Contoh *avarage pooling* :

Misalkan diambil matriks X pada Contoh Proses *Batch Normalization* sebagai matriks *input*. Kemudian tentukan ukuran filter *avarage pooling* dan *stride* yang digunakan. Sebagai contoh diambil filter *avarage pooling* berukuran 2×2 dan *stride* 1, sehingga matriks *input* X akan dipartisi menjadi sub-sub matriks yang berukuran 2×2 dengan pergesaran *stride* secara horizontal maupun vertikal sebesar 1. Adapun proses partisi sub-sub matriks dapat dilihat sebagai berikut ini:

$$X = \begin{bmatrix} 6{,}1 & 0 & 10{,}1 \\ 10{,}1 & 0 & 0 \\ 5{,}1 & 7{,}1 & 0 \end{bmatrix}$$

$$N_{11} = \text{avg}(G_{11}) = 4{,}05 \qquad N_{12} = \text{avg}(G_{12}) = 2{,}525$$

$$N_{21} = \text{avg}(G_{21}) = 5{,}575 \qquad N_{22} = \text{avg}(G_{22}) = 1{,}775$$

Dari keempat nilai tersebutpada masing-masing submatriks akan dimasukkan kedalam sebuah matriks N berukuran 2×2 yang merupakan hasil dari operasi *avarage pooling*. Matriks M tersebut dapat dilihat sebagai berikut:

$$N = \begin{bmatrix} 4{,}05 & 2{,}525 \\ 5{,}575 & 1{,}775 \end{bmatrix}$$

9. Global Avarage Pooling

Global avarage pooling berfungsi untuk menghitung nilai rata-rata matriks pada hasil *convolution* sebelumnya. Sebagai contoh diambil filter *global avarage pooling* $berukuran\ 2 \times 2$ dan *stride* 1, sehingga matriks *input* M dapat dilihat sebagai berikut :

$$N = \begin{bmatrix} 4{,}05 & 2{,}525 \\ 5{,}575 & 1{,}775 \end{bmatrix}$$

$$L = \left[\frac{4{,}05 + 2{,}525 + 5{,}575 + 1{,}775}{4}\right]$$

$$L = [3{,}48]$$

10. *Concatenate*

Concatenate digunakan untuk menggabungkan dua matriks *input* menjadi satu buah matriks *input* baru yang memiliki ukuran dimensi fitur yang berbeda. Proses penggabungan menggunakan *concatenate* dapat dilihat pada contoh berikut:

Contoh proses *concatenate* :

Misalkan diambil matriks V berukuran 2×4

$$V = \begin{bmatrix} 1 & 10 & 3 & 5 \\ 17 & 2 & 7 & 4 \end{bmatrix}$$

Dan matriks B berukuran 2×4

$$B = \begin{bmatrix} 10 & 2 & 3 & 9 \\ 5 & 11 & 6 & 8 \end{bmatrix}$$

Gabungkan matriks V dan matriks *B* dengan menggunakan *concatenate*, sehingga akan menghasilkan matriks *output D* dengan ukuran yang baru, proses tersebut dapat dilihat pada Gambar 1.

$$V = \begin{bmatrix} 1 & 10 & 3 & 5 \\ 17 & 2 & 7 & 4 \end{bmatrix}$$

$$B = \begin{bmatrix} 10 & 2 & 3 & 9 \\ 5 & 11 & 6 & 8 \end{bmatrix}$$

concatenate

$$D = \begin{bmatrix} 1 & 10 & 3 & 5 \\ 17 & 2 & 7 & 4 \\ 10 & 2 & 3 & 9 \\ 5 & 11 & 6 & 8 \end{bmatrix}$$

Pada proses penggabungan dengan menggunakan *concatenate*, dimana matriks *V* yang berukuran 2×4 digabungkan dengan matriks *B* berukuran 2×4 menghasilkan matriks *D* sebagai hasil matriks *concatenate* yang berukuran 4×4.

11. Fungsi aktivasi *sigmoid*

Perhitungan fungsi aktivasi *sigmoid* dengan menggunakan Persamaan (2.5).

Contoh perhitungan fungsi aktivasi sigmoid :

Misalkan diambil contoh matriks *Z* berukuran 3×3 yaitu:

$$Z = \begin{bmatrix} 6{,}1 & -2{,}9 & 10{,}1 \\ 10{,}1 & -1{,}9 & -5{,}9 \\ 5{,}1 & 7{,}1 & -2{,}9 \end{bmatrix}$$

Entri matriks Z pada baris 2 kolom 2 yaitu $z_{2,2} = -1{,}9$ Kemudian entri matriks $z_{2,2}$ disubsitusikan kedalam fungsi aktivasi *sigmoid* $\sigma(z)$ menjadi :

$$\sigma(-1{,}9) = \frac{1}{1 + e^{-(-1{,}9)}}$$

$$= \frac{1}{1 + e^{1{,}9}}$$

$$= \frac{1}{1 + 6{,}686}$$

$$= \frac{1}{7{,}686}$$

$$= 0{,}13$$

Lakukan hal yang sama untuk entri matriks Z lainnya sehingga memperoleh matriks *P* hasil dari subsitusi ke dalam fungsi aktivasi *sigmoid* sebagai berikut:

$$P = \begin{bmatrix} 0{,}997 & 0{,}052 & 0{,}999 \\ 0{,}999 & 0{,}130 & 0{,}002 \\ 0{,}993 & 0{,}999 & 0{,}052 \end{bmatrix}$$

12. *Softmax*

Fungsi aktivasi *softmax* yang digunakan pada *output layer* terakhir yang berfungsi untuk menghitung probablilitas dalam klasifikasi.

Contoh perhitungan fungsi aktivasi softmax :

Misalkan $z_1 = 3{,}25$ adalah bobot yang diberikan oleh sistem untuk arsitektur *InceptionV3*, $z_2 = 4{,}45$ adalah bobot yang diberikan oleh sistem untuk arsitektur *MobileNet*, dan $z_3 = 2{,}25$ adalah bobot yang diberikan oleh sistem untuk arsitektur VGG19. Kemudian substusi nilai z_i kedalam fungsi aktivasi *softmax* sebagai berikut:

$$p(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

$$p(z_1) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$= \frac{e^{3{,}25}}{e^{3{,}25} + e^{4{,}45} + e^{2{,}25}}$$

$$= \frac{25{,}79}{25{,}79 + 85{,}62 + 9{,}48}$$

$$= 0{,}213$$

$$p(z_2) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$= \frac{e^{4.45}}{e^{3.25} + e^{4.45} + e^{2.25}}$$

$$= \frac{85{,}62}{25.79 + 85.62 + 9.48}$$

$$= 0.71$$

$$p(z_3) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$= \frac{e^{2.25}}{e^{3.25} + e^{4.45} + e^{2.25}}$$

$$= \frac{9{,}48}{25.79 + 85.62 + 9.48}$$

$$= 0.078$$

13. *Weight voting*

Weight voting digunakan untuk mengambil keputusan prediksi dari bobot yang dihasilkan model.

Contoh perhitungan Weight voting :

Misalkan $w_1 = 0.25$ adalah bobot hasil *training* untuk arsitektur *InceptionV3*, $w_2 = 0.65$ adalah bobot hasil *training* untuk arsitektur *MobileNet*, dan $w_3 = 0.75$ adalah bobot hasil *training* untuk arsitektur VGG19. Untuk $p(z_i)$, ambil nilai fungsi aktivasi *softmax* pada Contoh perhitungan *Weight voting* . Lalu lakukan perhitungan nilai prediksi menggunakan Persamaan (2.14) sebagai berikut:

$$f(z_i) = \sum_{i=1}^{n} w_i p(z_i)$$

$$f(z_i) = \big(w_1 \cdot p(z_1)\big) + \big(w_2 \cdot p(z_2)\big) + \big(w_3 \cdot p(z_3)\big)$$

$$= (0.25 \cdot 0.213) + (0.65 \cdot 0.71) + (0.75 \cdot 0.078)$$

$$= 0.053 + 0.46 + 0.0585$$

$$=0{,}571$$

Berdasarkan hasil dari perhitungan pada Contoh perhitungan *weight voting* menghasilkan nilai bobot 0,571 dimana $0{,}5 \leq x \leq 1$ maka *output* bobot adalah 1 yang berarti DR, sedangkan *input* bobot $0 \leq x < 0{,}5$ maka *output* bobot adalah 0 yang berarti normal, maka hasil prediksi adalah 1 yg berarti DR.

14. *Loss Function Binary Cross Entropy*

Loss function binary cross entropy digunakan mengukur nilai kerugian yang diperoleh dalam proses *training.* Adapun untuk menghitung *loss function binary cross entropy* dapat dilihat pada Contoh perhitungan *loss function binary cross entropy* sebagi berikut:

Contoh perhitungan *loss fuction binary cross entropy*

Dimana n menunjukkan banyaknya data. y adalah hasil prediksi dimana bernilai 1 untuk label DR dan 0 untuk normal. p_i adalah nilai hasil prediksi sebenarnya. L adalah *loss function binary cross entropy.* Ambil $n = 3$, label ke sebenarnya atau $q_i = 1$ dan hasil dari fungsi aktivasi *softmax* atau p_k yang telah dihitung pada Subbab 4.4.12. Untuk menghitung *loss function binary cross entropy* dapat digunakan Persamaan (2.11).

$$\mathcal{L}(p,q) = -\frac{1}{n}\sum_{k=0}^{n}(q_k \log p_k + (1-q_k)\log(1-p_k))$$

$$= -\frac{1}{3}\sum_{k=1}^{3}(q_k \log p_k + (1-q_k)\log(1-p_k))$$

$$= -\frac{1}{3}(1 \log 0.213 + (1-1)\log(1-0.213))$$

$$+ (1 \log 0.71 + (1-1)\log(1-0.71)) + (1 \log 0.078$$

$$+ (1-1)\log(1-0.078))$$

$$= -\frac{1}{3}\left(-0.67 + (-0.148) + (-1.107)\right)$$

$$= -\frac{1}{3}(-1.925)$$

$$= 0.64$$

4.5. Proses Klasifikasi

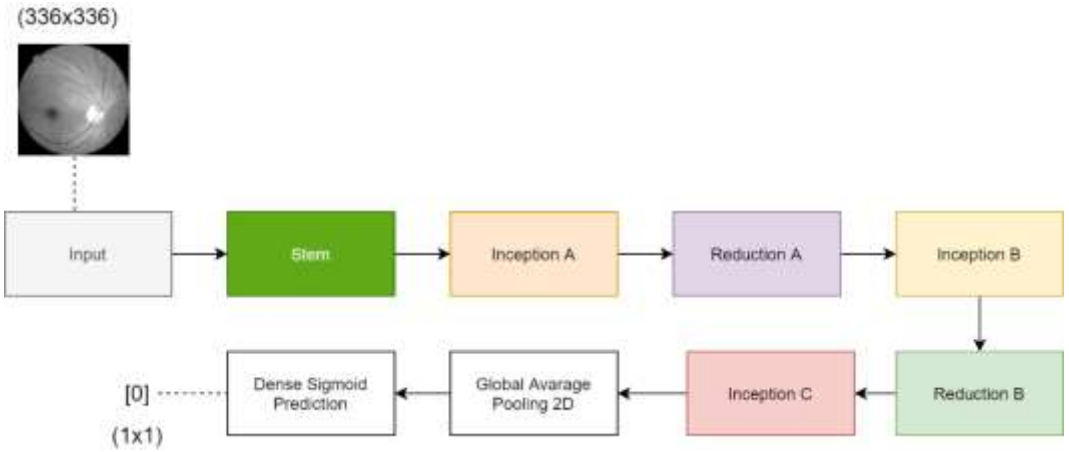
Dataset yang telah di perbanyak melalui augmentasi berjumlah 18000 data. Dataset tersebut dibagi menjadi dua kategori yaitu 16000 data *training* dan 2000 data *testing*. Pada penelitian ini akan menggunakan *MobileNet*, *InceptionV3*, VGG19, dan *Ensemble learning*.

1. *InceptionV3*

Pada *InceptionV3* akan dilakukan *training* data, *testing* data, dan evaluasi. Berikut ini penjelasannya :

a. *Training Data :*

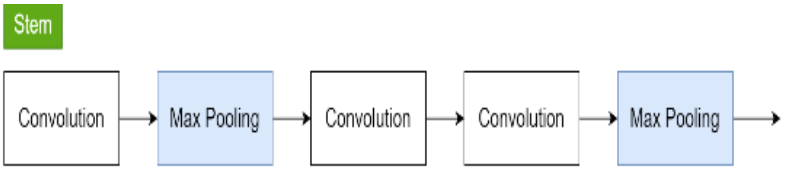
Data yang digunakan berjumlah 20 dari data *training* STARE. Data *training* tersebut telah dilakukan *pre-processing* dan augmentasi seperti pada subbab 4.2 dan 4.3. jumlah seluruh data *training* dari proses augmentasi berjumlah 16000 yang akan dibagi menjadi dua yaitu data *training* dan validasi sebanyak 14000 dan 2000. Dalam penelitian ini menggunakan jumlah pengulangan (*epoch*) pada *training* sebanyak 30, dan iterasi dalam satu *epoch* sebanyak 2468 iterasi. Tahapan ini dilakukan untuk melatih data menggunakan arsitektur yang dapat dilihat pada Gambar 4.1 untuk mengenali fitur-fitur dari data citra yang diberikan.



Gambar 4. 1 Ilustrasi arsitektur *InceptionV3* yang diusulkan

Dapat dilihat pada Gambar 4.5 terdapat 9 blok yaitu *Input*, Stem, *Inception* A, *Reduction* A, *Inception* B, *Reduction* B, *Inception* C, *Global avarage pooling* 2D, dan *Dense Sigmoid Prediction*. Adapun langkah-langkah sebagai berikut :

1. Pada blok yang pertama adalah *input* citra hasil *pre-processing* dan augmentasi dengan ukuran 336×336.
2. Pada blok Stem dilakukan beberapa tahapan yang dapat dilihat pada Gambar 4.2.

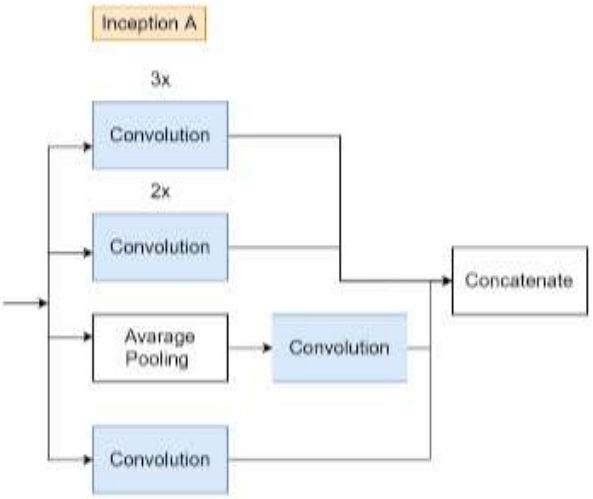


Gambar 4. 2 Tahapan Stem

Dapat dilihat dari Gambar 4.2 tahap awal yang akan dilakukan adalah *convolution layer*s seperti perhitungan pada Subbab 4.4.2 dan 4.4.3, selanjutnya dilakukannya *max pooling* sesuai perhitungan pada Subbab 4.4.7, selanjutnya dilakukan *convolution layer* lagi sebanyak 2 kali seperti

perhitungan pada Subbab 4.4.2 dan 4.4.3, dan terakhir di *max pooling* kembali sesuai perhitungan pada Subbab 4.4.7.

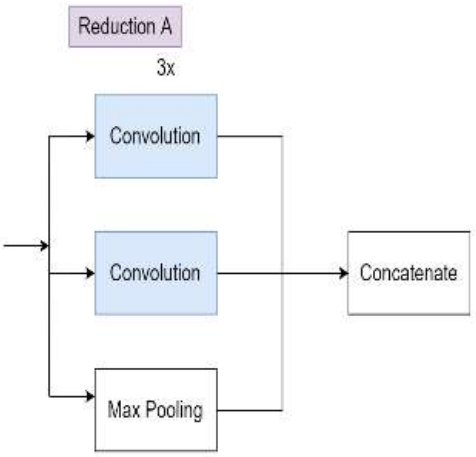
3. Pada langkah ini dilakukan proses pada blok *Inception* A terdiri dari 4 jalur yang dapat dilihat pada Gambar 4.3.



Gambar 4. 3 Tahapan *Inception* A

Pada jalur pertama dilakukan tiga kali *convolution layer* sesuai perhitungan pada Subbab 4.4.2 dan 4.4.3, jalur kedua dilakukan dua kali *convolution layer* sesuai perhitungan pada Subbab 4.4.2 dan 4.4.3, jalur ketiga dilakukan *avarage pooling* sesuai perhitungan pada Subbab 4.4.8 dan *convolution layer* sesuai perhitungan pada Subbab 4.4.2 dan 4.4.3, jalur keempat dilakukan *convolution layer* sesuai perhitungan pada Subbab 4.4.2 dan 4.4.3. Setiap jalur akan diakhiri dengan *concantened layer* sesuai perhitungan pada Subbab 4.4.10.

4. Pada blok *Reduction* A terdiri dari tiga jalur yang dapat dilihat pada Gambar 4.4.



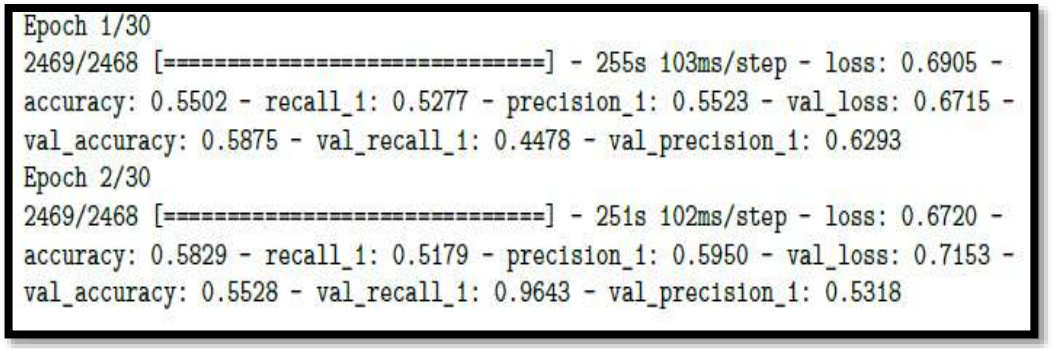
Gambar 4. 4 Tahapan *Reduction* A

Jalur pertama data akan dilakukan *convolution layer* sebanyak 3 kali sesuai perhitungan pada Subbab 4.4.2 dan 4.4.3, lalu jalur kedua dilakukan *convolution* sesuai perhitungan pada Subbab 4.4.2 dan 4.4.3, pada jalur ketiga akan dilakukan *max pooling* sesuai perhitungan pada Subbab 4.4.7. Setiap jalur akan diakhiri dengan *concantened layer* sesuai perhitungan pada Subbab 4.4.10. Pada blok *reduction* B yang membedakan hanya jumlah *convolution layer*, pada jalur pertama akan dilakukan *convolution layer* sebanyak 4 kali dan pada jalur kedua akan dilakukan *convolution layer* sebanyak 2 kali.

5. Pada blok *Inception* B pengerjaannya sama seperti pada langkah 3, yang membedakan hanya jumlah pengulangan pada jalur pertama dan jalur kedua.
6. Pada blok *Reduction* B proses pengerjaanya sama seperti langkah 4.
7. Pada blok *Inception* C pengerjaannya sama seperti pada langkah 3, yang membedakan hanya jumlah pengulangan pada jalur pertama dan jalur kedua.

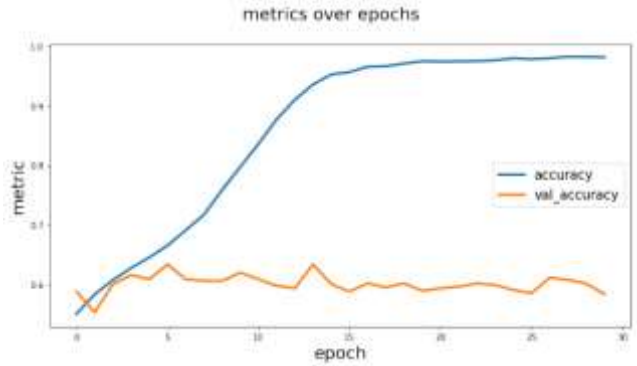
8. Selanjutnya dilakukan *global avarage pooling* seperti pada Subbab 4.4.9 dan fungsi aktivasi *sigmoid* sesuai dengan perhitungan pada Subbab 4.4.11.
9 Setelah langkah (5) dilakukan, gunakan *global avarege pooling* sesuai perhitungan pada Subbab 4.4.9 dan fungsi *sigmoid* sesuai perhitungan pada Subbab 4.4.11.
10 Kemudian dihitung nilai *loss* (*error*) untuk setiap *epoch* dan akurasi untuk data *training* sesuai perhitungan pada Subbab 4.4.14.
11 Lakukan penyimpanan bobot. Bobot akan disimpan jika nilai *error* pada data validasi lebih kecil dari *epoch* sebelumnya, jika tidak maka bobot akan diperbarui untuk *epoch* selanjutnya.
12 Ulangi langkah yang sama mulai dari langkah (2) sampai (6) hingga *epoch* terakhir.
13 Setelah proses *training* data dilakukan, lalu simpan model dan nilai *output* dari model.

Hasil proses *training* pada model InceptionV3 dapat dilihat pada Gambar 4.5 berikut.



Gambar 4. 5 Hasil *Training* data pada model *InceptionV3*

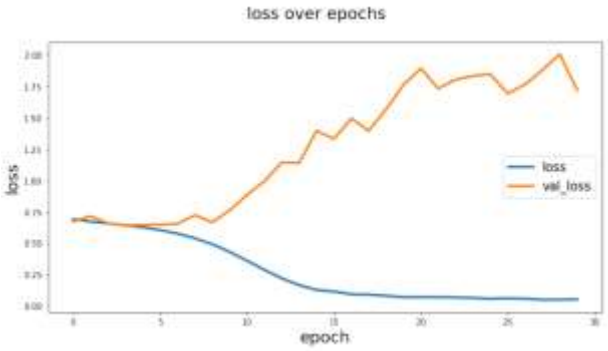
Pada *epoch* pertama menghasilkan akurasi sebesar 0,5502 dengan nilai *loss* sebesar 0,6905 untuk data latih, sedangkan untuk data valisasi menghasilkan nilai akurasi sebesar 0,5875 dengan nilai *loss* sebesar 0,6715. Berdasarkan nilai *loss* validasi yang dihasilkan, bobot akan disimpan dan digunakan untuk *epoch* kedua. Pada *epoch* ke-2 menghasilkan nilai akurasi sebesar 0,5829 dan nilai *loss* sebesar 0,6720 pada data *training*, sedangkan pada data validasi menghasilkan nilai akurasi sebesar 0,5528 dan nilai *loss* sebesar 0,7153. Berdasarkan nilai *loss* pada data validasi bobot tidak disimpan karena nilai *loss* lebih besar dari *epoch* sebelumnya. Nilai *loss* validasi akan terus diperbarui seiring hasil nilai *loss* validasi yang lebih baik. Proses *training* model arsitektur *InceptionV3* akan dilakukan hingga *epoch* ke-30 untuk memperoleh bobot terbaik yang akan digunakan untuk proses *testing*. Adapun grafik akurasi untuk data latih dan data validasi pada model *InceptionV3* yang didapatkan selama proses *training* dapat dilihat pada Gambar 4.6 sebagai berikut :



Gambar 4. 6 Grafik nilai akurasi pada proses *training* model *InceptionV3*

Pada Gambar 4.6 terlihat bahwa grafik akurasi model *InceptionV3* untuk data latih yang diperoleh selama proses *training* selalu meningkat pada setiap *epoch*. Pada *epoch* pertama akurasi yang diperoleh sebesar 0,5502 kemudian pada *epoch*

selanjutnya nilai akurasi meningkat terus menerus menuju 0,995. Nilai akurasi untuk data validasi pada *epoch* pertama sebesar 0,5875, kemudian mengalami penurunan kemudian pada *epoch* ke-2 nilai akurasinya menjadi 0,5528. Pada *epoch* ke-3 dan terus mengalami naik dan turun pada *epoch* selanjutnya hingga akhir *epoch* menghasilkan nilai akurasi utuk data validasi 0,58. Pada grafik validasi *accuracy* mengalami *overfitting* dimana seharusnya grafik validasi *accuracy* mengalami peningkatan. Kemudian diberikan juga grafik *loss* yang diperoleh selama proses *training* yang dapat dilihat pada Gambar 4.7.



Gambar 4. 7 Grafik nilai *loss* pada proses *training* model *InceptionV3*

Pada Gambar 4.7 terlihat bahwa grafik nilai *loss* arsitektur *InceptionV3* untuk data latih selama proses *training* selalu menurun pada setiap *epoch*. *Epoch* pertama menghasilkan nilai *loss* sebesar 0,6905, *epoch* ke-2 sampai dengan *epoch* ke-30 grafik selalu menurun menuju 0,20. Pada data validasi nilai *epoch* mengalami peningkatan dari *epoch* pertama hingga *epoch* ke-30 dengan nilai akhir *loss* sebesar 0,74. Grafik validasi *loss* mengalami *overfitting* dimana seharusnya grafik validasi *loss* mengalami penurunan. Melihat dari grafik hasil pada Gambar 4.6 dan Gambar 4.7 arsitektur *InceptionV3* yang diusulkan mengalami *overfitting*.

b. *Testing* dan Evaluasi

Pada tahap ini akan dilakukan pengujian menggunakan data *testing* untuk melihat keberhasilan model dalam melakukan klasifikasi citra. Pada tahap ini menghasilkan *confusion matrix* pada data *testing* yang dapat dilihat pada Tabel 4.4.

Tabel 4. 4 Hasil *confusion matrix* pada data *testing InceptionV3*.

Actual Values	*Prediction Values*	
	Normal	*Diabetic Retinopathy*
Normal	976	*24*
Diabetic Retinopathy	*173*	*827*

Tabel 4.4 dapat digunakkan untuk menghitung evaluasi arsitektur dilakukan dengan cara membandingkan nilai prediksi dan nilai aktual menggunakan *confusion matrix*. Hasil *confusion matrix* pada klasifikasi dapat dilihat pada Berdasarkan Tabel 4.4 diperoleh hasil bahwa model dapat menebak citra normal dengan benar sebanyak 976 citra, citra normal namun ditebak DR sebanyak 24 citra, citra DR ditebak normal sebanyak 173 citra, dan benar menebak citra DR sebanyak 827 citra. *Confusion matrix* tersebut dapat digunakan untuk menghitung kinerja arsitektur berdasarkan akurasi, sensitivitas, spesifisitas, *F1 Score* dan *Cohen's kappa* yang dapat dihitung menggunakan Persamaan (2.12) sampai dengan (2.16) berdasarkan *confusion matrix* pada Tabel 4.4 sebagai berikut :

1. Akurasi

$$Akurasi = \frac{TP + TN}{TP + TN + FP + FN} \times 100\%$$

$$= \frac{976 + 827}{976 + 827 + 173 + 24} \times 100\%$$

$$= \frac{1803}{2000} \times 100\%$$

$$= 90{,}15\%$$

2. Sensitivitas

$$\text{Sensitivitas} = \frac{TP}{TP + FN} \times 100\%$$

$$= \frac{976}{976 + 24} \times 100\%$$

$$= 97{,}6\%$$

3. Spesitifitas

$$\text{Spesifisitas} = \frac{TN}{FP + TN} \times 100\%$$

$$= \frac{827}{173 + 827} \times 100\%$$

$$= 82{,}7\%$$

4. F1-score

$$F1 - Score = \frac{2TP}{2TP + FP + FN} \times 100\%$$

$$= \frac{2(976)}{2(976) + 173 + 24} \times 100\%$$

$$= 90\%$$

5. *Cohen's kappa*

$$p_0 = \frac{TP + TN}{TP + TN + FP + FN}$$

$$= \frac{976 + 827}{976 + 827 + 173 + 24}$$

$$= 0{,}9015$$

$$p_e = \frac{\frac{(TP+FN)\times(TP+FP)}{TP+FN+FP+TN} + \frac{(FP+TN)\times(FN+TN)}{TP+FN+FP+TN}}{TP + FN + FP + TN}$$

$$= \frac{\frac{(976+24)\times(976+173)}{976+827+173+24} + \frac{(173+827)\times(24+827)}{976+827+173+24}}{976 + 827 + 173 + 24}$$

$$= \frac{574{,}5 + 425{,}5}{2000}$$

$$= 0{,}5$$

$$Cohen's\ kappa = \frac{p_0 - p_e}{1 - p_e}$$

$$= \frac{0{,}9015 - 0{,}5}{1 - 0{,}5}$$

$$= 0{,}80$$

Dari perhitungan diatas menghasilkan nilai akurasi sebesar 90,15%, sensitivitas sebesar 97,6%, nilai spesifisitas sebesar 82,7%, $F1\ Score$ sebesar 90% dan *Cohen's kappa* sebesar 0,80.

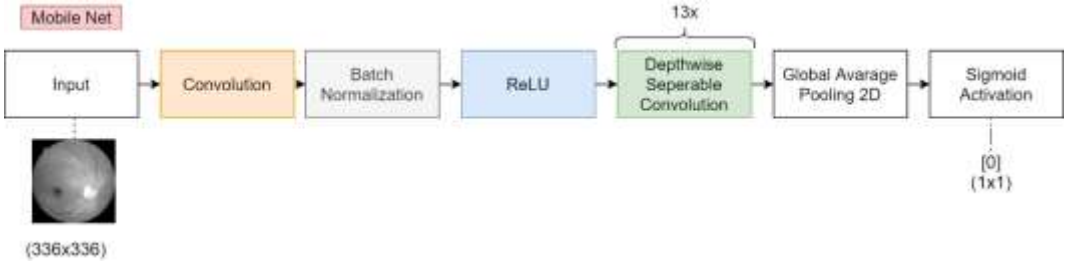
2. MobileNet

Pada *MobileNet* akan dilakukan *training* data, *testing* data, dan evaluasi data. Berikut ini penjelasannya :

a. *Training* Data

Data yang digunakan berjumlah 20 dari data *training* STARE. Data *training* tersebut telah dilakukan *pre-processing* dan augmentasi seperti pada subbab 4.2 dan 4.3. jumlah seluruh data *training* dari proses augmentasi sebanyak 16000 yang akan dibagi menjadi dua yaitu data *training* dan validasi sebanyak 13400 dan 1600. Dalam penelitian ini menggunakan jumlah pengulangan (*epoch*) pada *training*

sebanyak 30, dan iterasi dalam satu *epoch* sebanyak 2468 iterasi. Pada tahapan ini dilakukan untuk melatih data menggunakan arsitektur yang dapat dilihat pada Gambar 4.8 untuk mengenali fitur-fitur dari data citra yang diberikan.



Gambar 4. 8 Arsitektur *MobileNet* yang diusulkan.

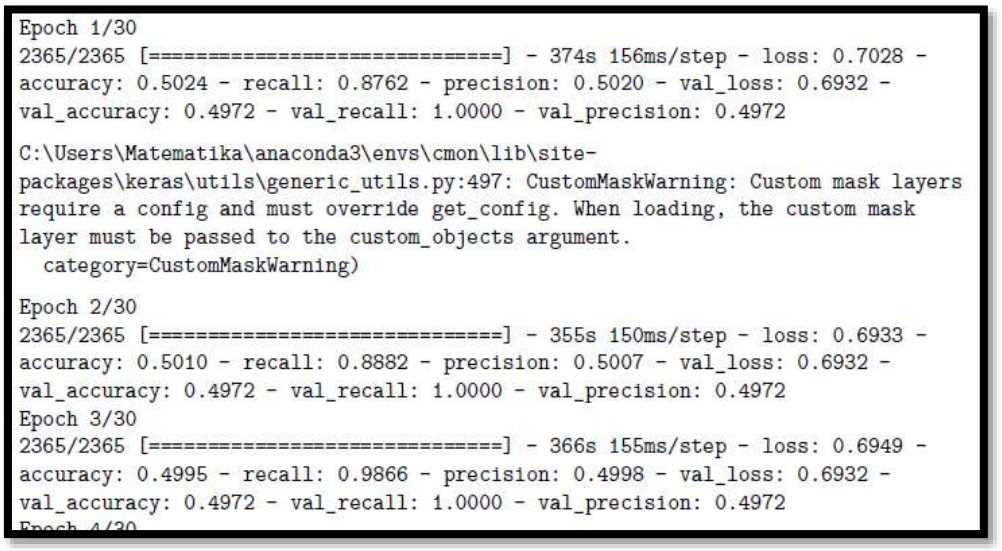
Adapun langkah-langkah pada Gambar 4.8 sebagai berikut:

1. Dilakukan pengaturan parameter seperti jumlah *epoch*, banyaknya data dalam satu *epoch* (*batch size*), nilai koreksi bobot (*learning rate*) dan *loss function* seperti perhitungan pada Subbab 4.4.14.
2. *Input* citra hasil augmentasi, selanjutnya lakukan proses *convolution layer* sesuai pada perhitungan pada Subbab 4.4.2 dan 4.4.3.
3. Setelah dilakukannya langkah (2), berikutnya lakukan *batch normalization* yang dapat dihitung sesuai pada perhitungan pada Subbab 4.4.5.
4. Kemudian menggunakan fungsi aktivasi ReLU yang dapat dihitung sesuai pada perhitungan pada Subbab 4.4.6.
5. Selanjutnya dilakukan proses *Depthwise Separable Convolutions* yang mana didalamnya terdapat *depthwise convolution* sesuai pada perhitungan pada Subbab 4.4.4, *batch normalization* yang dapat dilihat seperti langkah (3), fungsi aktivasi *ReLu* seperti langkah (4) , dan *pointwise convolution*

yang dapat dihitung sesuai pada perhitungan pada Subbab 4.4.4. Pada langkah ini akan diulang sebanyak 13 kali.

6. Setelah langkah (5) dilakukan, gunakan *global avarege pooling* sesuai pada perhitungan pada Subbab 4.4.9 dan fungsi *sigmoid* sesuai pada perhitungan pada Subbab 4.4.11.
7. Kemudian dihitung nilai *loss* (*error*) untuk setiap *epoch* dan akurasi untuk data *training* sesuai pada perhitungan pada Subbab 4.4.14.
8. Lakukan penyimpanan bobot. Bobot akan disimpan jika nilai *error* pada data validasi lebih kecil dari *epoch* sebelumnya, jika tidak maka bobot akan diperbarui untuk *epoch* selanjutnya.
9. Ulangi langkah yang sama mulai dari langkah (2) sampai (8) hingga *epoch* terakhir.
10. Setelah proses *training* data dilakukan, lalu simpan model dan nilai *output* dari model.

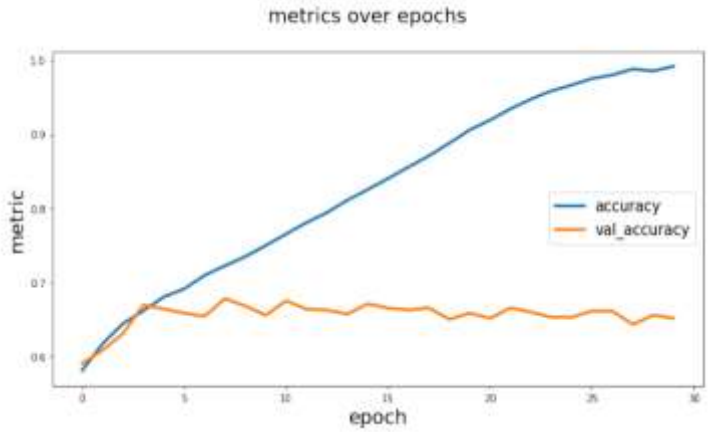
Hasil proses *training* pada model InceptionV3 dapat dilihat pada Gambar 4.9 berikut.



Gambar 4. 9 hasil *Training* data pada model *MobileNet*

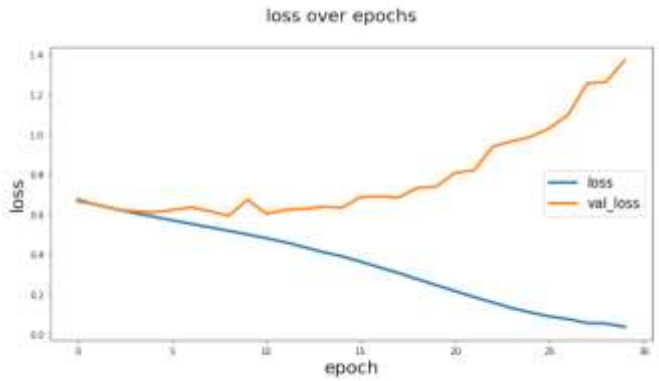
Hasil proses *training* pada *epoch* pertama menghasilkan akurasi sebesar 0,5236 dengan nilai *loss* sebesar 0.7123 untuk data latih, sedangkan untuk data valisasi menghasilkan nilai akurasi sebesar 0,5472 dengan nilai *loss* sebesar 0,6916. Berdasarkan nilai *loss* validasi yang dihasilkan, bobot akan disimpan dan digunakan untuk *epoch* kedua. Pada *epoch* ke-2 data *training* menghasilkan nila akurasi sebesar 0,5010 dan nilai *loss* sebesar 0,6933, sedangkan nilai akurasi pada data validasi sebesar 0,4972 dan nilai *loss* sebesar 0,6932. Berdasarkan nilai *loss* validasi yang dihasilkan, bobot akan disimpan dan digunakan untuk *epoch* ke-3. Pada *epoch* ke-3 diperoleh nilai akurasi pada data *training* sebesar 0,4995 dan nilai *loss* sebesar 0,6949, sedangkan pada data validasi diperoleh nilai akurasi sebesar 0,497 dan nilai *loss* sebesar 0,6932. Berdasarkan nilai *loss* pada data Nilai *loss* validasi akan terus diperbarui seiring hasil nilai *loss* validasi yang lebih baik. Proses *training* model arsitektur *MobileNet* akan dilakukan hingga *epoch* ke-30 untuk memperoleh bobot terbaik yang akan digunakan untuk proses *testing*. Adapun grafik akurasi untuk data

latih dan data validasi pada model *MobileNet* yang didapatkan selama proses *training* dapat dilihat pada Gambar 4.10 sebagai berikut :



Gambar 4. 10 Grafik nilai akurasi pada proses *training* model *MobileNet*

Pada Gambar 4.10 terlihat bahwa grafik akurasi model *MobileNet* untuk data latih yang diperoleh selama proses *training* selalu meningkat pada setiap *epoch*. *Epoch* pertama diperoleh nilai sebesar 0,5236 dan selalu meningkat pada tiap *epoch*, *epoch* ke-30 diperoleh nilai akurasi sebesar 0,98. Nilai akurasi untuk data validasi pada *epoch* pertama sebesar 0,5472, kemudian mengalami peningkatan hingga *epoch* ke-3 dan *epoch* ke-4 mengalami penurunan. Nilai akurasi pada data validasi mengalami kenaikan dan penurunan nilai akurasi hingga *epoch* ke-30 diperoleh hasil akurasi sebesar 0,6014. Pada grafik validasi mengalami *overfitting* dimana seharusnya grafik validasi mengalami peningkatan . Kemudian diberikan juga grafik *loss* yang diperoleh selama proses *training* yang dapat dilihat pada Gambar 4.11 sebagai berikut.



Gambar 4. 11 Grafik nilai *loss* pada proses *training* model *MobileNet*

Pada Gambar 4.11 terlihat bahwa grafik nilai *loss* arsitektur *MobileNet* untuk data latih selama proses *training* selalu menurun pada setiap *epoch*. Pada *epoch* pertama diperoleh nilai *loss* sebesar 0,7123, kemudian dari *epoch* ke-2 hingga *epoch* ke-30 nilai *loss* selalu menurun. Namun pada grafik validasi *loss* mengalami *overfitting* dimana seharusnya grafik validasi mengalami penurunan. Melihat dari grafik pada Gambar 4.10 dan Gambar 4.11 pada arsitektur *MobileNet* yang diusulkan mengalami *overfitting*.

b. *Testing* dan Evaluasi

Berdasarkan tabel 2.1 untuk menghitung evaluasi arsitektur dapat dilakukan dengan cara membandingkan nilai prediksi dan nilai aktual menggunakan *confusion matrix*. Hasil *confusion matrix* pada klasifikasi menggunakan arsitektur *MobileNet* dapat dilihat pada Tabel 4.5 berikut.

Tabel 4. 5 Hasil *confusion matrix* data *testing* pada *MobileNet*.

Actual Values	*Prediction Values*	
	Normal	*Diabetic Retinopathy*
Normal	923	*77*
Diabetic Retinopathy	*450*	*550*

Dari Tabel 4.5 diperoleh hasil bahwa model dapat menebak citra normal dengan benar sebanyak 923 citra, citra normal namun ditebak DR sebanyak 77 citra, citra DR ditebak normal sebanyak 450 citra, dan benar menebak citra DR sebanyak 550 citra. *Confusion matrix* tersebut dapat digunakan untuk menghitung kinerja arsitektur berdasarkan akurasi, sensitivitas, spesifisitas, dan menggunakan Persamaan (2.12) sampai dengan (2.16) berdasarkan *confusion matrix* pada Tabel 4.5 sebagai berikut :

1. Akurasi

$$Akurasi = \frac{TP + TN}{TP + TN + FP + FN} \times 100\%$$

$$= \frac{923 + 550}{923 + 550 + 450 + 77} \times 100\%$$

$$= \frac{1473}{2000} \times 100\%$$

$$= 73{,}65\%$$

2. Sensitivitas

$$\text{Sensitivitas} = \frac{TP}{TP + FN} \times 100\%$$

$$= \frac{923}{923 + 77} \times 100\%$$

$$= 92{,}3\%$$

3. Spesitifitas

$$\text{Spesifisitas} = \frac{TN}{FP + TN} \times 100\%$$

$$= \frac{550}{450 + 550} \times 100\%$$

$$= 55\%$$

4. F1-score

$$F1 - \ Score = \frac{2TP}{2TP + FP + FN} \times 100\%$$

$$= \frac{2(923)}{2(923) + 450 + 77} \times 100\%$$

$$= 77{,}7\%$$

5. *Cohen's kappa*

$$p_0 = \frac{TP + TN}{TP + TN + FP + FN}$$

$$= \frac{923 + 550}{923 + 550 + 450 + 77}$$

$$= 0{,}7365$$

$$p_e = \frac{\frac{(TP+FN)\times(TP+FP)}{TP+FN+FP+TN} + \frac{(FP+TN)\times(FN+TN)}{TP+FN+FP+TN}}{TP + FN + FP + TN}$$

$$= \frac{\frac{(923+77)\times(923+450)}{923+550+450+77} + \frac{(450+550)\times(77+550)}{923+550+450+77}}{923 + 550 + 450 + 77}$$

$$= \frac{686{,}5 + 263{,}5}{2000}$$

$$= 0{,}5$$

$$Cohen's\ kappa = \frac{p_0 - p_e}{1 - p_e}$$

$$= \frac{0{,}7365 - 0{,}5}{1 - 0{,}5}$$

$$= 0{,}473$$

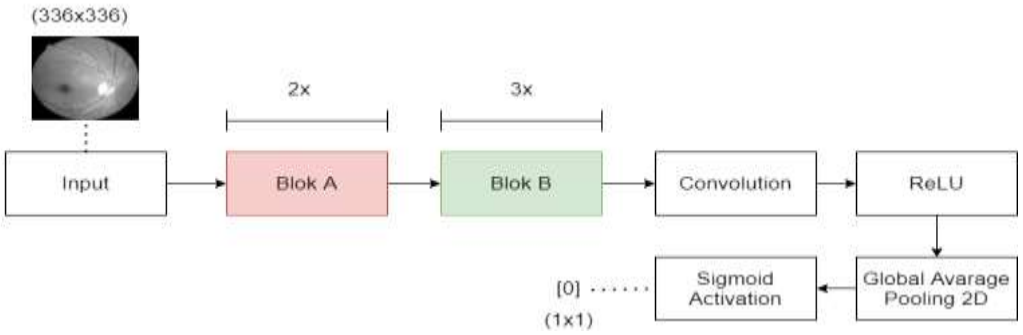
Dari perhitungan tersebut menghasilkan nilai akurasi sebesar 73,65%, nilai sensitivitas sebesar 92,3%, nilai spesitifitas sebesar 55%, nilai *F1-Score* sebesar 77,7%, dan nilai *Cohen's kappa* sebesar 0,473.

3. VGG19

Pada VGG19 akan dilakukan *training* data, *testing* data, dan evaluasi. Berikut ini penjelasannya :

a. *Training* :

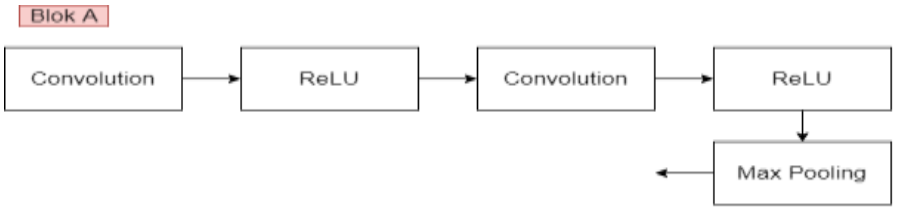
Data yang digunakan berjumlah 20 dari data *training* STARE. Data *training* tersebut telah dilakukan *pre-processing* dan augmentasi seperti pada subbab 4.2 dan 4.3. jumlah seluruh data *training* dari proses augmentasi berjumlah 16000 yang akan dibagi menjadi dua yaitu data *training* dan validasi sebanyak 14400 dan 1600. Dalam penelitian ini menggunakan jumlah pengulangan (*epoch*) pada *training* sebanyak 30, dan iterasi dalam satu *epoch* sebanyak 2468 iterasi. Tahapan ini dilakukan untuk melatih data menggunakan arsitektur yang dapat dilihat pada Gambar 4.12 untuk mengenali fitur-fitur dari data citra yang diberikan.



Gambar 4. 12 Ilustrasi arsitektur VGG19 yang diusulkan

Langkah-langkah pada Gambar 4.12 akan dijelaskan sebagai berikut :

1. *Input* data *training* kemudian lakukan proses pada residual blok A sebanyak 2 kali, yang mana blok tersebut dapat dilihat pada Gambar 4.13.



Gambar 4. 13 Isi dari blok A pada arsitektur VGG19

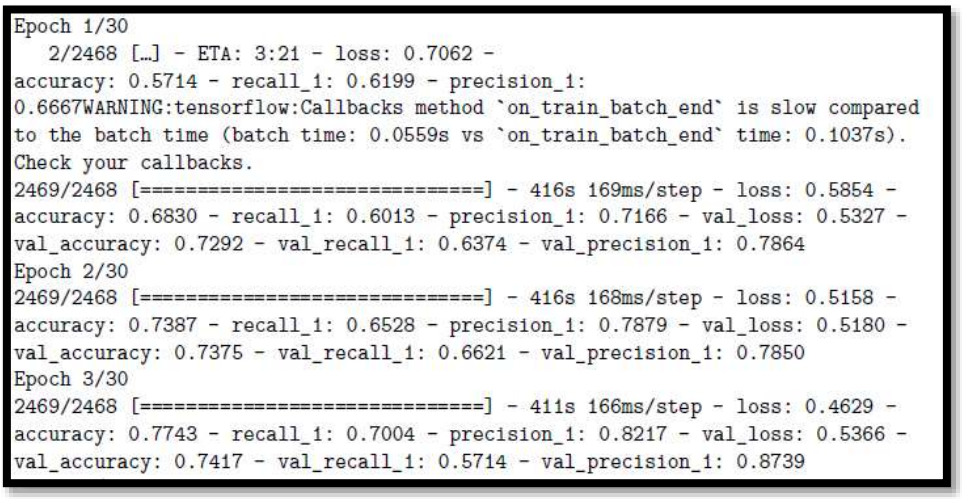
Blok A terdiri dari 5 blok yang pertama dilakukan *convolution layer* sesuai perhitungan pada Subbab 4.4.2 dan 4.4.3. setelah dilakukan *convolution layer* dilanjutkan dengan Fungsi aktivasi *ReLU* sesuai perhitungan pada Subbab 4.4.6, ulangi kembali *convolution layer* dan ReLU, dan terakhir dilakukan prose *max pooling* sesuai perhitungan pada Subbab 4.4.7.

2. Kemudian proses pada *residual* blok B sebanyak 3 kali, dimana isi dari blok B sama seperti langkah (1).
3. Setelah langkah (2) dilanjukan dengan *convolution layer* sesuai perhitungan pada Subbab 4.4.2 dan 4.4.3, fungsi aktivasi *ReLu* sesuai

perhitungan pada Subbab 4.4.6.

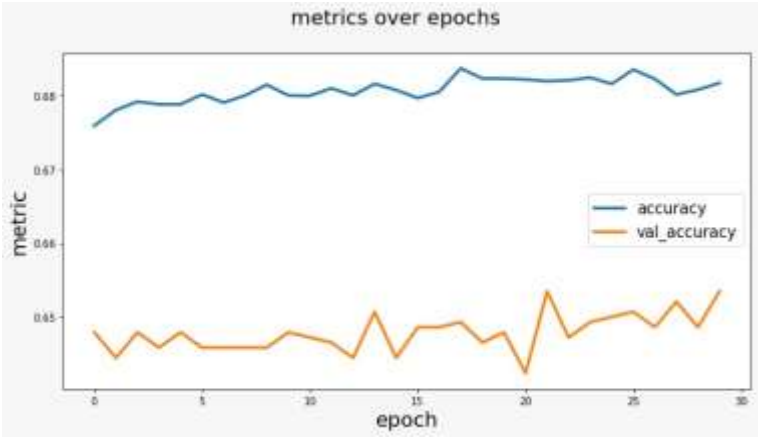
4. Dari langkah (3) dilakukan *global avarage pooling* sesuai perhitungan pada Subbab 4.4.9 , dan fungsi aktivasi *sigmoid* sesuai perhitungan pada Subbab 4.4.11.
5 Setelah langkah (4) dilakukan, gunakan *global avarege pooling* sesuai perhitungan pada Subbab 4.4.9 dan fungsi *sigmoid* sesuai perhitungan pada Subbab 4.4.11.
6 Hitung nilai *loss* (*error*) untuk setiap *epoch* dan akurasi untuk data *training* seperti pada Subbab 4.4.14.
7 Lakukan penyimpanan bobot. Bobot akan disimpan jika nilai *error* pada data validasi lebih kecil dari *epoch* sebelumnya, jika tidak maka bobot akan diperbarui untuk *epoch* selanjutnya.
8 Ulangi langkah yang sama mulai dari langkah (1) sampai (7) hingga *epoch* terakhir.
9 Setelah proses *training* data dilakukan, lalu simpan model dan nilai *output* dari model.

Hasil proses *training* pada model VGG19 dapat dilihat pada Gambar 4.14 berikut.



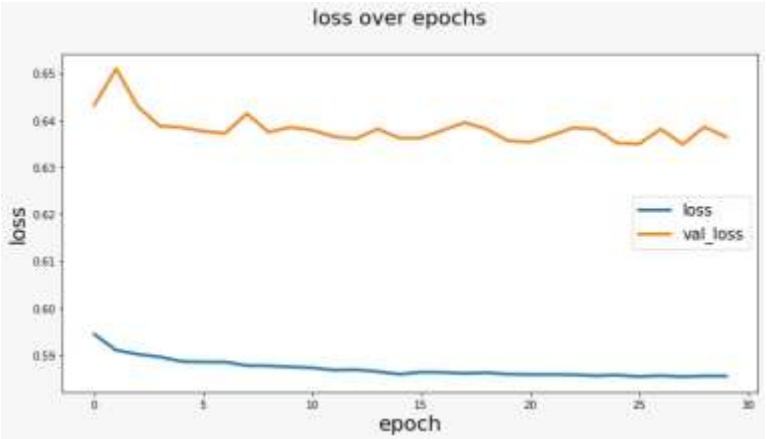
Gambar 4. 14 hasil Training pada model VGG19

Hasil proses *training* pada *epoch* pertama menghasilkan akurasi sebesar 0,6830 dengan nilai *loss* sebesar 0.5854 untuk data latih, sedangkan untuk data valisasi menghasilkan nilai akurasi sebesar 0,7292 dengan nilai *loss* sebesar 0,5327. Berdasarkan nilai *loss* validasi yang dihasilkan, bobot akan disimpan dan digunakan untuk *epoch* ke-2. Pada *epoch* ke-2 menghasilkan akurasi sebesar 0,7387 dengan nilai *loss* sebesar 0.5158 untuk data latih, sedangkan untuk data valisasi menghasilkan nilai akurasi sebesar 0,7375 dengan nilai *loss* sebesar 0,5180. Berdasarkan nilai *loss* validasi yang dihasilkan, bobot akan disimpan dan digunakan untuk *epoch* ke-3. Nilai *loss* validasi akan terus diperbarui seiring hasil nilai *loss* validasi yang lebih baik. Proses *training* model arsitektur VGG19 akan dilakukan hingga *epoch* ke-30 untuk memperoleh bobot terbaik yang akan digunakan untuk proses *testing*. Adapun grafik akurasi untuk data latih dan data validasi pada model VGG19 yang didapatkan selama proses *training* dapat dilihat pada Gambar 4.15 sebagai berikut :



Gambar 4. 15 Grafik nilai akurasi pada proses *training* model VGG19

Pada Gambar 4.15 terlihat bahwa grafik akurasi model VGG19 untuk data latih yang diperoleh selama proses *training* mengalami peningkatan dan penurunan pada setiap *epoch*. Pada *epoch* pertama akurasi yang diperoleh sebesar 0,6830 kemudian pada *epoch* selanjutnya nilai akurasi meningkat terus menerus menuju 0,995. Pada grafik validasi *accuracy* mengalami *overfitting* dimana seharusnya grafik validasi mengalami peningkatan. Kemudian diberikan juga grafik *loss* yang diperoleh selama proses *training* yang dapat dilihat pada Gambar 4.16 sebagai berikut.



Gambar 4. 16 Grafik nilai *loss* pada proses *training* model VGG19

Pada Gambar 4.16 terlihat bahwa grafik nilai *loss* arsitektur VGG19 untuk data latih selama proses *training* selalu menurun pada setiap *epoch*. Namun pada grafik validasi *loss* mengalami *overfitting* dimana seharusnya grafik validasi accuracy mengalami penurunan. Melihat dari grafik pada Gambar 4.15 dan Gambar 4.16 pada arsitektur VGG19 yang diusulkan mengalami *overfitting*.

b. *Testing* dan Evaluasi

Pada tahap ini akan dilakukan pengujian menggunakan data *testing* untuk melihat keberhasilan model dalam melakukan klasifikasi citra. Berdasarkan tabel 2.1 untuk menghitung evaluasi arsitektur dapat dilakukan dengan cara membandingkan nilai prediksi dan nilai aktual menggunakan *confusion matrix*. Hasil *confusion matrix* pada klasifikasi dapat dilihat pada Tabel 4.6 berikut.

Tabel 4. 6 Hasil *confusion matrix* data *testing*.

Actual Values	*Prediction Values*	
	Normal	*Diabetic Retinopathy*
Normal	1000	*0*
Diabetic Retinopathy	*225*	*775*

Berdasarkan Tabel 4.6 diperoleh hasil bahwa model dapat menebak citra normal dengan benar sebanyak 1000 citra, citra DR ditebak normal sebanyak 225 citra, dan benar menebak citra DR sebanyak 775 citra. *Confusion matrix* tersebut dapat digunakan untuk menghitung kinerja arsitektur berdasarkan akurasi, sensitivitas, spesifisitas, dan menggunakan Persamaan (2.12) sampai dengan (2.16) berdasarkan *confusion matrix* pada Tabel 4.6 sebagai berikut :

1. Akurasi

$$Akurasi = \frac{TP + TN}{TP + TN + FP + FN} \times 100\%$$

$$= \frac{1000 + 775}{1000 + 775 + 225 + 0} \times 100\%$$

$$= \frac{1775}{2000} \times 100\%$$

$$= 88{,}75\%$$

2. Sensitivitas

$$\text{Sensitivitas} = \frac{TP}{TP + FN} \times 100\%$$

$$= \frac{1000}{1000 + 0} \times 100\%$$

$$= 100\%$$

3. Spesitifitas

$$\text{Spesifisitas} = \frac{TN}{FP + TN} \times 100\%$$

$$= \frac{775}{225 + 775} \times 100\%$$

$$= 77{,}5\%$$

4. F1-score

$$F1 - Score = \frac{2TP}{2TP + FP + FN} \times 100\%$$

$$= \frac{2(1000)}{2(1000) + 225 + 0} \times 100\%$$

$$= 89{,}9\%$$

5. *Cohen's kappa*

$$p_0 = \frac{TP + TN}{TP + TN + FP + FN}$$

$$= \frac{1000 + 775}{1000 + 775 + 225 + 0}$$

$$= 0{,}8875$$

$$p_e = \frac{\frac{(TP+FN)\times(TP+FP)}{TP+FN+FP+TN} + \frac{(FP+TN)\times(FN+TN)}{TP+FN+FP+TN}}{TP + FN + FP + TN}$$

$$= \frac{\frac{(1000+0)\times(1000+225)}{1000+775+225+0} + \frac{(0+775)\times(225+775)}{1000+775+225+0}}{1000 + 775 + 225 + 0}$$

$$= \frac{612{,}5 + 387.5}{2000}$$

$$= 0{,}5$$

$$Cohen's\ kappa = \frac{p_0 - p_e}{1 - p_e}$$

$$= \frac{0{,}8875 - 0{,}5}{1 - 0{,}5}$$

$$= 0{,}77$$

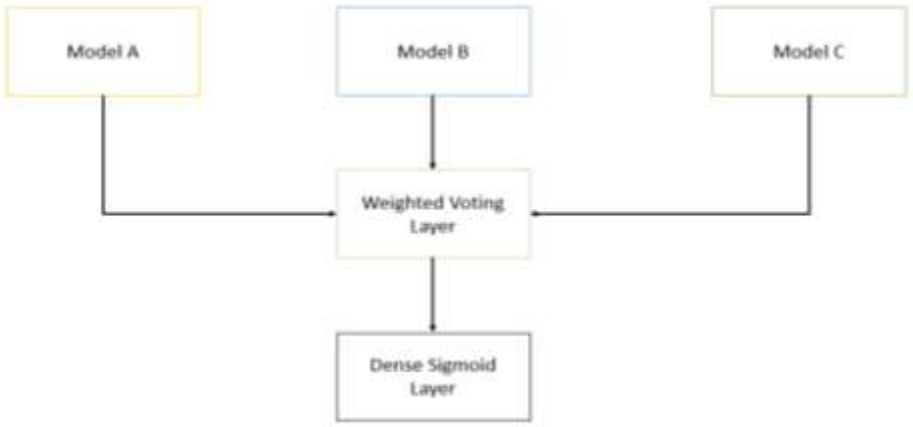
Pada perhitungan manual *confusion matrix* menghasilkan nilai akurasi sebesar 88,75%, sensitivitas sebesar 100%, nilai spesifisitas sebesar 77,5% , $F1 - Score$ sebesar 89,8% dan *Cohen's kappa* sebesar 0,77.

4. *Ensemble learning*

Pada *Ensemble learning* akan dilakukan *training* data, *testing* data, dan evaluasi. Berikut ini penjelasannya :

a. *Training*

Pada tahap ini dilakukan pengambilan dari bobot akhir tiap arsitektur, kemudian penelitian ini menggunakan jumlah pengulangan (*epoch*) pada *training* sebanyak 50, dan iterasi dalam satu *epoch* sebanyak 600 iterasi. Tahapan ini dilakukan untuk melatih data menggunakan arsitektur. Proses *training* data pada *Ensemble learning* dapat dilihat pada Gambar 4.17.



Gambar 4. 17 Ilustrasi *Ensemble learning*

Berikut penjelasan dari Gambar 4.17.

1. Pada gambar terdapat 3 model yang dimana model A (*InceptionV3*), model B (*MobileNet*) dan model C (VGG19) yang akan diambil bobot akhir dari masing-masing model tanpa mengubah isi dari masing-masing model tersebut.
2. Bobot akhir dari masing-masing model akan digabungkan menggunakan *weighted voting layers* seperti pada Subbab 4.4.13.
3. Terakhir akan dilakukan *dense sigmoid layer* seperti pada perhitungan pada Subbab 4.4.11.

4. Setelah langkah (4) dilakukan, gunakan *global avarege pooling* seperti pada Subbab 4.4.8 dan fungsi *sigmoid* seperti pada Subbab 4.4.11.
5. Kemudian dihitung nilai *loss* (*error*) untuk setiap *epoch* dan akurasi untuk data *training* seperti pada Subbab 4.4.14.
6. Lakukan penyimpanan bobot. Bobot akan disimpan jika nilai *error* pada data validasi lebih kecil dari *epoch* sebelumnya, jika tidak maka bobot akan diperbarui untuk *epoch* selanjutnya.
7. Ulangi langkah yang sama mulai dari langkah (1) sampai (7) hingga *epoch* terakhir.
8. Setelah proses *training* data dilakukan, lalu simpan model dan nilai *output* dari model.

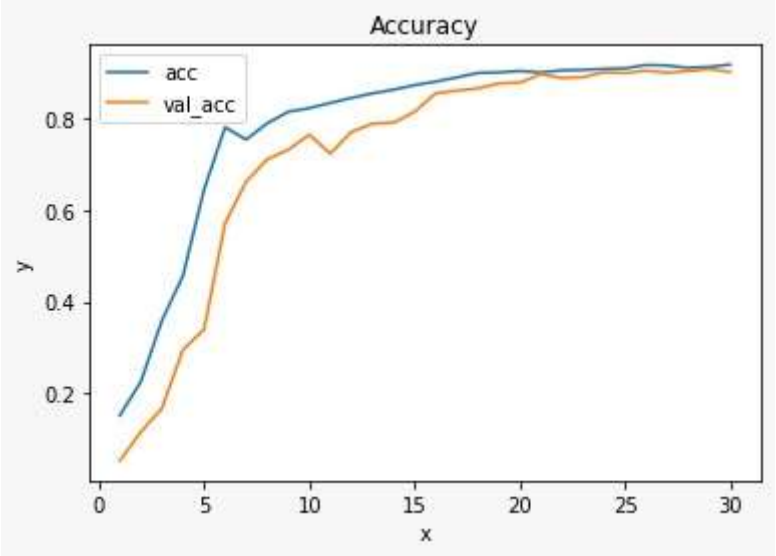
   Hasil proses *training* pada model *Ensemble* learning dapat dilihat pada Gambar 4.18 berikut.

```
Epoch 1/50
600/600 [==============================] - 347s 549ms/step - loss: 0.8944 -
accuracy: 0.1005 - recall: 0.0000e+00 - precision: 0.0000e+00 - val_loss: 0.7461
- val_accuracy: 0.3930 - val_recall: 0.4750 - val_precision: 0.4081

C:\Users\Matematika\anaconda3\envs\cmon\lib\site-
packages\keras\utils\generic_utils.py:497: CustomMaskWarning: Custom mask layers
require a config and must override get_config. When loading, the custom mask
layer must be passed to the custom_objects argument.
  category=CustomMaskWarning)

Epoch 2/50
600/600 [==============================] - 327s 545ms/step - loss: 0.8472 -
accuracy: 0.0161 - recall: 0.0000e+00 - precision: 0.0000e+00 - val_loss: 0.7325
- val_accuracy: 0.3740 - val_recall: 0.4880 - val_precision: 0.3974
Epoch 3/50
600/600 [==============================] - 327s 545ms/step - loss: 0.8055 -
accuracy: 0.0030 - recall: 0.0000e+00 - precision: 0.0000e+00 - val_loss: 0.7212
- val_accuracy: 0.3805 - val_recall: 0.5740 - val_precision: 0.4138
```

Gambar 4. 18 Hasil *Training* Data pada Model *Ensemble Learning*

Dari Gambar 4.18 terlihat nilai *epoch* pertama pada model *Ensemble* learning

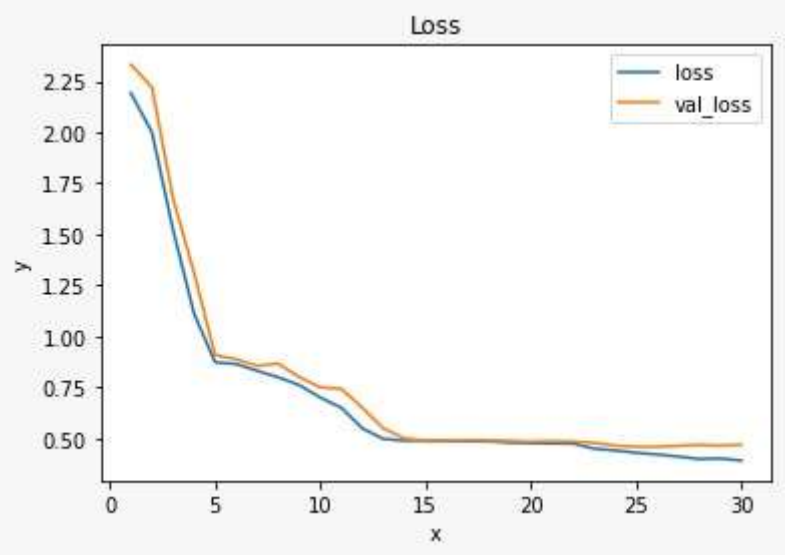
diperoleh nilai akurasi pada data *training* sebesar 0,1005 dan nilai *loss* sebesar 0,8944, sedangkan pada data validasi diperoleh nilai akurasi sebesar 0,3930 dan nilai *loss* sebesar 0,7461. Berdasarkan nilai *loss* pada data validasi, bobot akan disimpan dan digunakan pada *epoch* ke-2. Pada *epoch* ke-2 diperoleh nilai akurasi pada data *training* sebesar 0,061 dan nilai *loss* sebesar 0,8472, sedangkan pada data validasi diperoleh nilai akurasi sebesar 0,3740 dan nilai *loss* 0,7325. Berdasarkan nilai *loss* pada data validasi maka bobot akan disimpan dan digunakan pada *epoch* ke-3.Adapun grafik akurasi untuk data *training* dan data validasi pada model *Ensemble* dapat dilihat pada Gambar 4.19 sebagai berikut:



Gambar 4. 19 Grafik nilai akurasi pada proses *training* model *Ensemble learning*

Pada Gambar 4.19 terlihat bahwa grafik akurasi model *Ensemble learning* untuk data latih yang diperoleh selama proses *training* selalu meningkat pada setiap *epoch*. Pada *epoch* pertama akurasi yang diperoleh sebesar 0,5236 kemudian pada *epoch* selanjutnya nilai akurasi meningkat terus menerus menuju 0,995. Nilai akurasi untuk data validasi awalnya mengalami penurunan kemudian pada *epoch*

ke-6 kemudian naik pada *epoch* ke-8 dan terus meningkat. Kemudian diberikan juga grafik *loss* yang diperoleh selama proses *training* yang dapat dilihat pada Gambar 4.19 sebagai berikut.



Gambar 4. 20 Grafik nilai *loss* pada proses *training* model *Ensemble* learning

Pada Gambar 4.20 terlihat bahwa grafik nilai *loss* dan validasi *loss* arsitektur *Ensemble learning* untuk data latih selama proses *training* selalu menurun pada setiap *epoch*.

b. *Testing* dan Evaluasi

Pada tahap ini akan dilakukan pengujian menggunakan data *testing* untuk melihat keberhasilan model dalam melakukan klasifikasi citra. Berdasarkan tabel 2.1 untuk menghitung evaluasi arsitektur dapat dilakukan dengan cara membandingkan nilai prediksi dan nilai aktual menggunakan *confusion matrix*. Hasil *confusion matrix* pada klasifikasi dapat dilihat pada Tabel 4.7 berikut.

Tabel 4. 7 Hasil *confusion matrix* data *testing*.

Actual Values	*Prediction Values*	
	Normal	*Diabetic Retinopathy*
Normal	954	*46*
Diabetic Retinopathy	*43*	*957*

Berdasarkan Tabel 4.7 diperoleh hasil bahwa model dapat menebak citra normal dengan benar sebanyak 954 citra, citra normal namun ditebak DR sebanyak 46 citra,citra DR ditebak normal sebanyak 43 citra, dan benar menebak citra DR sebanyak 957 citra. *Confusion matrix* tersebut dapat digunakan untuk menghitung kinerja arsitektur berdasarkan akurasi, sensitivitas, spesifisitas, dan menggunakan Persamaan (2.12) sampai dengan (2.16) berdasarkan *confusion matrix* pada Tabel 4.4 sebagai berikut :

1. Akurasi

$$Akurasi = \frac{TP + TN}{TP + TN + FP + FN} \times 100\%$$

$$= \frac{1911}{2000} \times 100\%$$

$$= 95{,}5\%$$

2. Sensitivitas

$$\text{Sensitivitas} = \frac{TP}{TP + FN} \times 100\%$$

$$= \frac{954}{954 + 46} \times 100\%$$

$$= 95{,}4\%$$

3. Spesitifitas

$$\text{Spesifisitas} = \frac{TN}{FP + TN} \times 100\%$$

$$= \frac{957}{43 + 957} \times 100\%$$

$$= 95{,}7\%$$

4. *F1-Score*

$$F1 - \ Score = \frac{2TP}{2TP + FP + FN} \times 100\%$$

$$= \frac{2(954)}{2(954) + 43 + 46} \times 100\%$$

$$= 95{,}5\%$$

5. *Cohen's kappa*

$$p_0 = \frac{TP + TN}{TP + TN + FP + FN}$$

$$= \frac{954 + 957}{954 + 957 + 43 + 46}$$

$$= 0{,}9015$$

$$p_e = \frac{\frac{(TP+FN)\times(TP+FP)}{TP+FN+FP+TN} + \frac{(FP+TN)\times(FN+TN)}{TP+FN+FP+TN}}{TP + FN + FP + TN}$$

$$= \frac{\frac{(954+46)\times(954+43)}{954+46+43+957} + \frac{(43+957)\times(46+957)}{954+46+43+957}}{954 + 46 + 43 + 957}$$

$$= \frac{498{,}5 + 501{,}5}{2000}$$

$$= 0{,}5$$

$$Cohen's\ kappa = \frac{p_0 - p_e}{1 - p_e}$$

$$= \frac{0{,}9555 - 0{,}5}{1 - 0{,}5}$$

$$= 0{,}911$$

Pada perhitungan manual *confusion matrix* menghasilkan nilai akurasi sebesar 95,5%, sensitivitas sebesar 95,4%, nilai spesifisitas sebesar 95,7% , $F1 - Score$ sebesar 95,5% dan nilai *Cohen's kappa* sebesar 0,911 .

4.5 Analisis dan Interpretasi Hasil

Pada penelitian ini telah dilakukan proses klasifikasi *diabetic retinopati* menggunakan arsitektur *MobileNet*, *InceptionV3*, VGG 19, dan *Ensemble learning*. Dengan menggunakan 20 citra STARE kemudian dilakukan proses augmentasi yang menghasilkan 18000 data baru, kemudian data tersebut dibagi menjadi data *training* dan data *testing*. Pada data *training* akan dibagi lagi menjadi dua yaitu data *training* sebanyak 14000 dan data validasi sebanyak 2000 data. Pada masing-masing proses *training* data diperoleh 2 grafik yaitu grafik nilai akurasi dan grafik nilai *loss*. Jika dilihat dari grafik nilai akurasi dan grafik nilai *loss* pada model klasifikasi tunggal mengalami *overfitting* pada nilai grafik validasi, namun setelah dilakukan *Ensemble* grafik nilai akurasi dan grafik nilai *loss* tidak lagi mengalami *overfitting*.

Tahap selanjutnya dilakukan *testing* data dengan 2000 data. Pada proses *testing* pada masing-masing arsitektur tunggal dan *Ensemble* diperoleh *confusion matrix* yang kemudian dihitung untuk mengetahui kinerja dari model tersebut. Adapun perbandingan hasil nilai kinerja klasifikasi masing-masing arsitektur tunggal dengan *Ensemble learning* dapat dilihat pada Tabel 4.8 berikut :

Tabel 4. 8 Perbandingan hasil klasifikasi dari masing-masing model.

Metode	Akurasi (%)	Sensitivitas (%)	Spesifisitas (%)	F1 Score (%)	*Cohen's kappa*
MobileNet	73,65	92,3	55	77,7	0,473
InceptionV3	90,15	97,6	82,7	90	0,80
VGG19	88,75	**100**	77,5	89,9	0,77
Ensemble learning	**95,5**	95,4	**95,7**	**95,5**	**0,911**

Keterangan : Angka yang dicetak tebal adalah nilai persentase tertinggi.

Pada Tabel 4.8 diperoleh nilai akurasi yang sangat baik, namun nilai akurasi pada arsitektur *MobileNet* masih cukup baik. Kemampuan model dalam memprediksi citra DR dengan benar sudah sangat baik, nilai kinerja sensitivitas tertinggi diperoleh pada model VGG19. Kemampuan model dalam memprediksi citra normal dengan benar pada model *Ensemble* sudah sangat baik, namun pada model *MobileNet* masih sangat kurang. tertinggi diperoleh pada metode *Ensemble learning* sebesar 95,5%, sensitivitas tertinggi diperoleh pada VGG19 sebesar 100%, spesifisitas tertinggi diperoleh pada *Ensemble learning* sebesar 95.7%, *F1-Score* tertinggi diperoleh pada *Ensemble learning* sebesar 95,5%, dan *Cohen's kappa* sebesar 0,911. Perbandingan hasil klasifikasi penelitian ini dengan penelitian lain dapat dilihat pada Tabel 4.9 berikut :

Tabel 4. 9 Perbandingan arsitektur dengan penelitian lain

Metode	Akurasi (%)	Sensitivitas (%)	Spesifisitas (%)	F1 Score (%)	*Cohen's kappa*
(*MobileNet*) Patel (2020)	89	-	-	-	-
(InceptionV3 dan QIY model) Hagos and Kant (2019)	90	-	-	-	-
Ensemble learning (*InceptionV3*, Resnet152, inception-ResnetV2) Jiang *et al.* (2019)	88,21	85,57	88,41	-	-
Ensemble learning (VGG16 dan VGG19) Mishra *et al.* (2020)	83	80	82	-	-
VGG19 Kwasigroch *et al.* (2018)	81	89.5	50.5	-	0.776
Metode *Ensemble learning* yang diusulkan	**95,5**	**95,4**	**95,7**	**95,5**	**0,911**

Berdasarkan tabel 4.9 Model arsitektur yang diusulkan memperoleh nilai akurasi, sensitivitas, spesifisitas, F1-Score, dan *Cohen's kappa* tertinggi dari penelitian lainnya. Berdasarkan tabel 2.2 dan hasil evaluasi kinerja model arsitektur yang diusulkan dapat disimpulkan bahwa kekauratan suatu arsitektur untuk melakukan klasifikasi DR sudah sangat baik ditunjukkan oleh nilai akurasi sebesar 95,5%. Untuk nilai sensitivitas yang dihasilkan sebesar 95,4% juga sangat baik dibandingkan penelitian lain yang berarti kemampuan model dalam memprediksi

DR dengan benar sudah sangat baik. Untuk nilai spesitifitas sebesar 95,7% artinya kemampuan model dalam memprediksi citra normal dengan benar sudah sangat baik. Untuk nilai *F1-Score* yang diperoleh sebesar 95,5% artinya harmonisasi antara sensitivitas dan spesifisitas sudah sangat baik, dan nilai *Cohen's kappa* sebesar 0,911 yang dihasilkan sudah sangat baik yang artinya model telah konsisten.