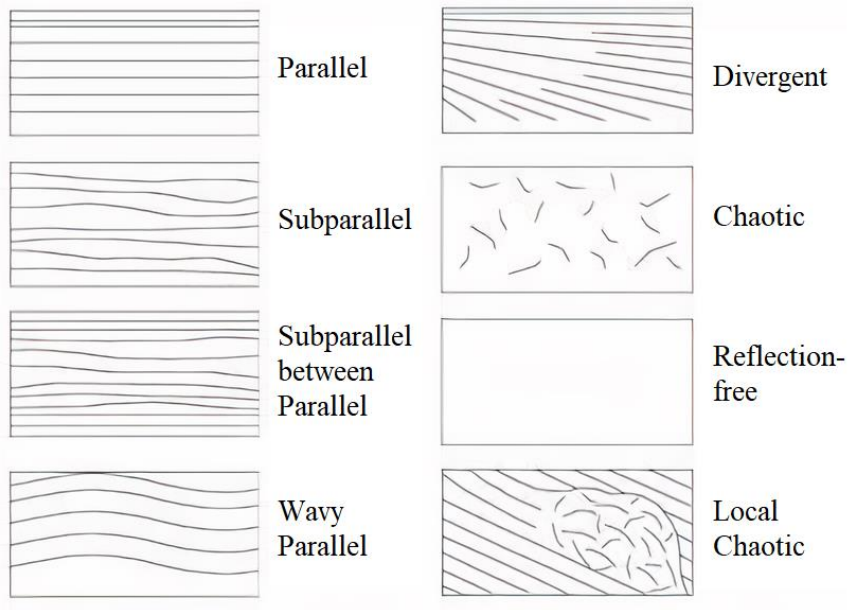
BAB II TEORI DASAR

2.1 Analisis Seismik Fasies

Analisis seismik fasies merupakan penafsiran fasies dari suatu lingkungan pengendapan berdasarkan data seismik, penafsiran ini mencakup dari geometri dan kesinambungan pola reflektor, amplitudo, frekuensi, dan terminasi refleksi dari bagian internal dan eksternal dari paket-paket reflektor tersebut (Mitchum dkk., 1997). Berikut ini diuraikan macam-macam konfigurasi internal seismik yang digunakan dalam interpretasi fasies seismik pengendapan, antara lain:

a. *Parallel*, dicirikan dengan pengendapan sedimen seragam atau terdapat di paparan (*shelf*) dengan subsiden yang seragam atau sedimentasi pada suatu *basin plain* yang stabil.
b. *Subparallel*, terjadi akibat situasi yang tidak tenang, terganggu oleh arus laut dan sistem pengendapan cepat pada zona pengisian.
c. *Subparallel between parallel*, terbentuk pada *fluvial plain* di mana endapan memiliki butiran sedang dengan kondisi tektonik yang stabil.
d. *Wavy Parallel*, terbentuk akibat lipatan kompresi dari lapisan *parallel* di atas permukaan *detachment* atau diapir atau *sheet drape* dengan endapan berbutir halus.
e. *Divergent,* terbentuk akibat permukaan yang miring secara progresif selama proses sedimentasi.
f. *Chaotic,* terbentuk akibat pengendapan dengan energi tinggi (*mounding, cut and fill channel*) atau deformasi setelah proses sedimentasi (sesar, gerakan *overpressure shale*, dll.)
g. *Reflection Free*, dapat ditemukan pada batuan beku, kubah garam, dan interior *reef* tunggal.
h. *Local Chaotic,* terjadi pada daerah *slump* (biasanya laut dalam) yang diakibatkan oleh gempa bumi atau ketidakstabilan gravitasi dan pengendapan terjadi dengan cepat.
i. *Konfigurasi Divergent (D)*, dicirikan oleh bentuk *wedge* di mana penebalan lateral lebih disebabkan penebalan dari refleksi itu sendiri bukan karena *onlap, toplap,* atau erosi.



Gambar 2.1 Tekstur internal seismik (Mitchum dkk., 1997).

2.2 *Machine Learning*

Machine learning atau pembelajaran mesin adalah teknik untuk memprediksi suatu data menggunakan pendekatan matematis (Putra, 2020). Dalam praktiknya *machine learning* digunakan untuk menggantikan atau meniru perilaku manusia dengan membuat model matematis yang merefleksikan pola-pola data sehingga dapat membantu manusia dalam menyelesaikan masalah atau melakukan otomatisasi (Tanaka, 2014). Dalam pembelajaran *machine learning*, terdapat dua skenario yaitu sebagai berikut:

1. *Supervised Learning*

   Penggunaan *supervised learning* menggunakan masukan data pembelajaran yang telah diberi label. Setelah itu membuat prediksi dari data yang telah diberi label tersebut. Umumnya *supervised learning* digunakan untuk melakukan klasifikasi.

2. *Unsupervised Learning*

   Penggunaan *unsupervised learning* menggunakan masukan data pembelajaran yang tidak diberi label. Kemudian dilakukan pengelompokan data berdasarkan karakteristik-karakteristik yang ditemui. Contoh permasalahan *unsupervised learning* adalah *clustering.*

2.2.1 *Training, Validation,* dan *Testing Set*

Dalam membangun model maupun mengevaluasi model dibutuhkan suatu kumpulan data (sampel dalam statistik) yang disebut sebagai *dataset. Dataset* dibagi menjadi tiga jenis, yaitu:

1. *Training set,* yaitu kumpulan data yang digunakan untuk melatih atau membangun model.
2. *Validation set,* yaitu kumpulan data yang digunakan untuk mengoptimisasi saat melatih model.
3. *Testing set,* yaitu kumpulan data yang digunakan untuk menguji model setelah proses *training* selesai.

Umumnya rasio pembagian *dataset* adalah (80% *training*: 10% *validation*: 10% *testing*) (Putra, 2020). Namun dalam beberapa kasus tertentu tidak terdapat *validation set.* Maka hal tersebut dapat diatasi dengan mengevaluasi model menggunakan metode *K-cross validation.*

2.2.2 Evaluasi Kinerja Model

Evaluasi kinerja model dapat dilakukan pada saat *training* dan dalam mengukur *performance measure* pada data *test.* Tujuannya untuk meminimalkan *cross-entropy* saat *training* sehingga diharap memberikan nilai probabilitas yang tinggi untuk kelas yang tepat dan nilai rendah untuk kelas lainnya serta mengkonversikannya menjadi keputusan final dengan memilih satu kelas yang tepat untuk *input* data yang diberikan (Putra, 2020). Adapun cara dalam mengevaluasi keputusan prediksi final model yaitu:

1. Akurasi

Akurasi didefinisikan sebagai proporsi prediksi yang benar dibagi dengan banyaknya sampel. Secara matematis, akurasi diformulasikan pada persamaan 2.1, dimana *N* merepresentasikan banyaknya sampel.

$$Akurasi = \frac{1}{N}\sum_{i=1}^{N} verdict_i$$
$$verdict_i = f(x) = \begin{cases} 1, & y_i = \widehat{y_i} \\ 0, & y_i \neq \widehat{y_i} \end{cases} \quad (2.1)$$

Sebagai contoh, suatu model pembelajaran mesin mengklasifikasikan gambar batuan menjadi "batu pasir" dan "batu lempung". *Desired output* y_i dan prediksi $\widehat{y_i}$ model diberikan pada tabel 2.1. Pada tabel ini, terdapat tujuh prediksi yang sama dengan *desired output*, yaitu sampel dengan *instance* = [1,3,5,7,8,9,10]. Sehingga akurasi model yang didapat yaitu $\frac{7}{10}$ = 0.7.

Tabel 2.1 Contoh hasil prediksi model.

Instance	***Desired Output***	**Hasil Prediksi**
1	Batu Pasir	Batu Pasir
2	Batu Pasir	Batu Lempung
3	Batu Lempung	Batu Lempung
4	Batu Pasir	Batu Lempung
5	Batu Lempung	Batu Lempung
6	Batu Lempung	Batu Pasir
7	Batu Lempung	Batu Lempung
8	Batu Pasir	Batu Pasir
9	Batu Pasir	Batu Pasir
10	Batu Lempung	Batu Lempung

2. F1 *Score*

F1 *score* adalah ukuran akurasi dari suatu tes dengan didefinisikan sebagai *harmonic mean* antara presisi dan *recall* kelas tersebut (Sokolova, 2006). Secara matematis F1 *score* dilambangkan dengan $F1^k$ dan dapat ditulis pada persamaan 2.2, dengan presisi pada persamaan 2.3, dan *recall* pada persamaan 2.4 dengan *k* sebagai kelas/kategori.

$$F1^k = 2\frac{Presisi^k \times Recall^k}{Presisi^k + Recall^k} \tag{2.2}$$

$$Presisi^k = \frac{Jumlah\ prediksi\ benar\ pada\ kelas\ k}{\|\hat{y}^k\|} \tag{2.3}$$

$$Recall^k = \frac{Jumlah\ prediksi\ benar\ pada\ kelas\ k}{\|y^k\|} \tag{2.4}$$

Untuk memudahkan perhitungan F1 *score*, presisi, dan *recall* maka dapat disusun menggunakan tabel *confusion matrix* prediksi yang diilustrasikan pada Tabel 2.2.

Tabel 2.2 *Confusion matrix* prediksi batu lempung dan batu pasir berdasarkan Tabel 2.1.

		Prediksi		$\|\boldsymbol{y^k}\|$
		Batu Lempung	Batu Pasir	
Desired	Batu Lempung	4	1	5
output	Batu Pasir	2	3	5
	$\|\hat{y}^k\|$	6	4	

Dari Tabel 2.2 maka dapat dihitung nilai dari presisi, *recall*, dan F1 *score* kategori batu lempung dan batu pasir yang ditunjukkan pada Tabel 2.3.

Tabel 2.3 Perhitungan nilai presisi, *recall*, dan F1 *score* untuk masing-masing kelas untuk contoh Tabel 2.1

Kategori	**Presisi**	***Recall***	**F1 *score***
Batu Lempung	$\frac{4}{6} = 0.67$	$\frac{4}{5} = 0.8$	$2\frac{0.67 \times 0.8}{0.67 + 0.8} = 0.73$
Batu Pasir	$\frac{3}{4} = 0.75$	$\frac{3}{5} = 0.6$	$2\frac{0.75 \times 0.6}{0.75 + 0.6} = 0.67$

Untuk mendapatkan nilai F1 secara keseluruhan terdapat dua cara, yaitu F1 *macro* dan F1 *weighted*. F1 *macro* dapat dilakukan dengan merata-ratakan F1 *score* disemua kelas. Secara matematis F1 *macro* dapat ditulis pada persamaan 2.5.

$$F1\ macro = \frac{1}{N}\sum_{k=1}^{K} F1^k \tag{2.5}$$

Sedangkan F1 *weighted* dapat dilakukan dengan memberikan bobot untuk masing-masing kelas, berdasarkan banyaknya *instance* yang seharusnya diklasifikasikan ke kelas tersebut. Secara matematis F1 *weighted* dapat ditulis pada persamaan 2.6.

$$F1\ weighted = \frac{1}{\sum_{k=1}^{K} y^k}\sum_{k=1}^{K} y^k F1^k \tag{2.6}$$

Berdasarkan contoh pada Tabel 2.1 maka nilai F1 *macro* dan F1 *weighted* adalah sebagai berikut:

$$F1\ macro = \frac{0.73 + 0.67}{2} = 0.7$$

$$F1\ weighted = \frac{5\ \times 0.73 + 5 \times 0.67}{10} = 0.7$$

2.2.3 *Multiclass Classification*

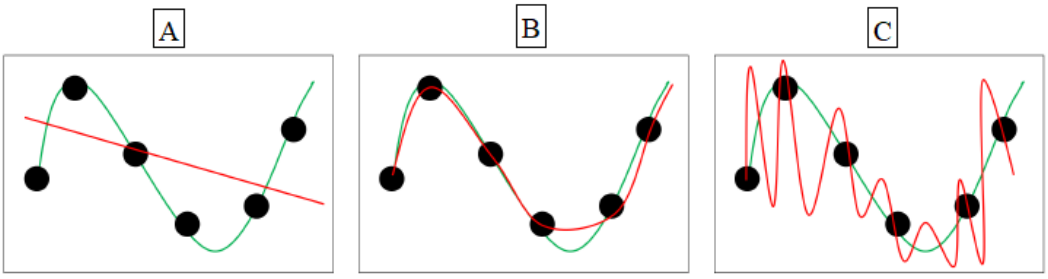
Multiclass classification yaitu suatu sampel yang diklasifikasikan ke dalam suatu himpunan kelas (Great Learning Team, 2020). Sampel tersebut hanya dapat berkorespondensi dengan satu kelas saja. Sehingga label pada *multiclass classification* bersifat *mutually exclusive*. Berikut disajikan Tabel 2.4 berupa contoh *multiclass classification.*

Tabel 2.4 Ilustrasi *multiclass classification*. Nilai "1" adalah *TRUE* sedangkan nilai "0" adalah *FALSE* (*class assignment*).

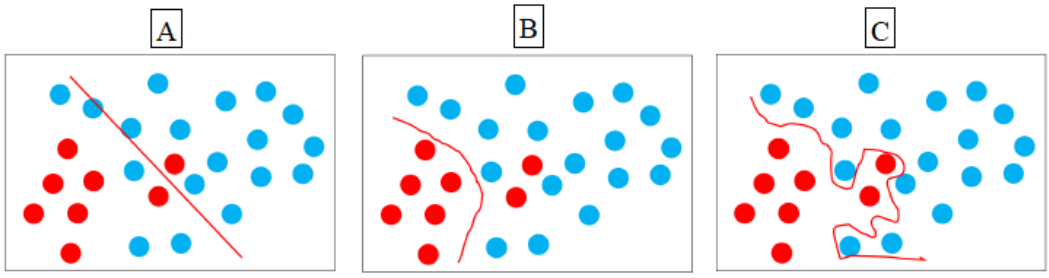
Instance	***Batu Breksi***	***Batu Pasir***	**Batu Lempung**
Gambar-1	1	0	0
Gambar-2	0	1	0
Gambar-3	0	0	1
Gambar-4	0	1	0

2.2.4 *Overfitting* dan *Underfitting*

Underfitting merupakan keadaan ketika kinerja model bernilai buruk pada keseluruhan *dataset* (Putra, 2020). Hal ini terjadi ketika model sangat tidak fleksibel. Model memiliki kemampuan yang rendah untuk mengestimasi variasi fungsi sehingga hasil prediksi sangat tidak bagus. Sedangkan *overfitting* merupakan keadaan ketika kinerja model bernilai baik untuk *training* tetapi buruk untuk *unseen data* (Putra, 2020). Hal ini terjadi ketika model terlalu fleksibel. Model memiliki kemampuan yang terlalu tinggi dalam mengestimasi banyak fungsi dan terlalu cocok terhadap *training set* sehingga hasil prediksi seolah sempurna. Berikut disajikan ilustrasi suatu kondisi data mengalami *underfitting* dan *overfitting* pada kasus regresi (Gambar 2.2) dan kasus klasifikasi (Gambar 2.3).



Gambar 2.2 Kondisi *underfitting* (A), normal (B), dan *overfitting* (C) pada regresi.

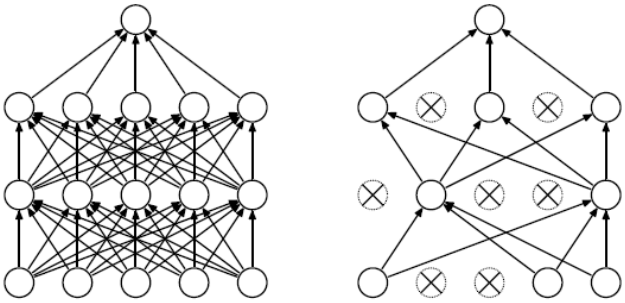


Gambar 2.3 Kondisi *underfitting* (A), normal (B), dan *overfitting* (C) pada klasifikasi.

2.2.5 *Regularization*

Regularization merupakan teknik yang digunakan untuk memodifikasi model *neural network* sehingga *generalization error* dapat diminimalisir (GoodFellow dkk., 2016). Adapun cara yang digunakan yaitu dengan memberikan *constraint* pada parameter. *Regularization* menjadi solusi untuk mengatasi permasalahan

model yang sensitif ataupun *overfitting* (Krizhevsky dkk., 2012). Salah satu jenis *regularization* yang dapat digunakan pada proses *training neural network* yaitu *dropout. Dropout* adalah sebuah teknik *regularization* dengan memberikan nilai *keep probability* untuk setiap *hidden layer* pada *neural network* dengan tujuan memilih beberapa neuron secara random dan tidak akan dipakai selama proses *training* (Srivastava dkk., 2014). Berikut disajikan ilustrasi proses *dropout* (Gambar 2.4).

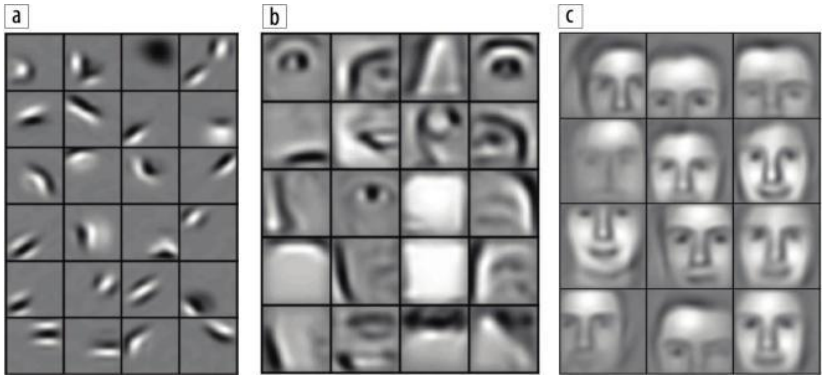


Gambar 2.4 *Dropout Regularization*. (Kiri) merupakan *neural network* dengan dua *hidden layer* sebelum dilakukan *dropout*. (Kanan) merupakan *neural network* yang telah diaplikasikan *dropout* (Srivastava et al, 2014).

Berdasarkan gambar tersebut terlihat bahwa beberapa neuron tidak terpakai lagi dalam proses *training* sehingga memiliki potensi untuk meningkatkan performa model dan mengurangi *overfitting.*

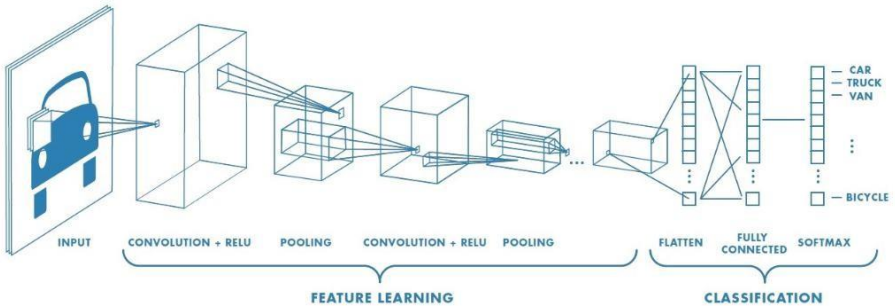
2.3 *Convolutional Neural Network*

Convolutional Neural Network (CNN) merupakan pengembangan dari *multilayer perceptron* (MLP) yang ditujukan untuk mengolah data dua dimensi dalam bentuk citra (Hubel dan Wiesel, 1968). Dari pengertian tersebut CNN termasuk kedalam jaringan syaraf buatan tingkat tinggi dan umum digunakan untuk mengolah data citra. Prinsip sederhana mengenai cara kerja CNN akan disajikan berupa kasus pendeteksi wajah manusia. Klasifikasi menggunakan CNN diselesaikan dengan membagi sebuah objek menjadi beberapa objek yang lebih kecil. Sebagai contoh, wajah terdiri dari dua mata, satu mulut, satu hidung, dan dua telinga. Hidung, mata, telinga, dan mulut dapat diuraikan menjadi fitur seperti tepi, kurva, lingkaran, atau gumpalan warna. CNN menggunakan fitur dari objek tersebut dengan terlebih dahulu mendeteksi fitur tingkat rendah kemudian menyatukannya untuk mendeteksi ke fitur yang lebih tinggi (Gambar 2.5).



Gambar 2.5 Visualisasi bagaimana wajah dapat direpresentasikan dengan CNN. (a) tepi dengan orientasi berbeda membentuk (b) mata, hidung, dan mulut yang bisa digunakan untuk mewakili (c) wajah (Lee et al., 2011).

Secara teknis, CNN adalah sebuah arsitektur yang dapat dilatih dan terdiri dari beberapa tahap. Input dan *output* dari setiap tahap terdiri dari beberapa *array* yang disebut dengan *feature map*. Setiap tahap terdiri dari tiga *layer* yaitu *convolution*, *activation function* dan *pooling layer*. Ilustrasi mengenai jaringan arsitektur CNN (Gambar 2.6).



Gambar 2.6 Arsitektur *Convolutional Neural Network*[1]

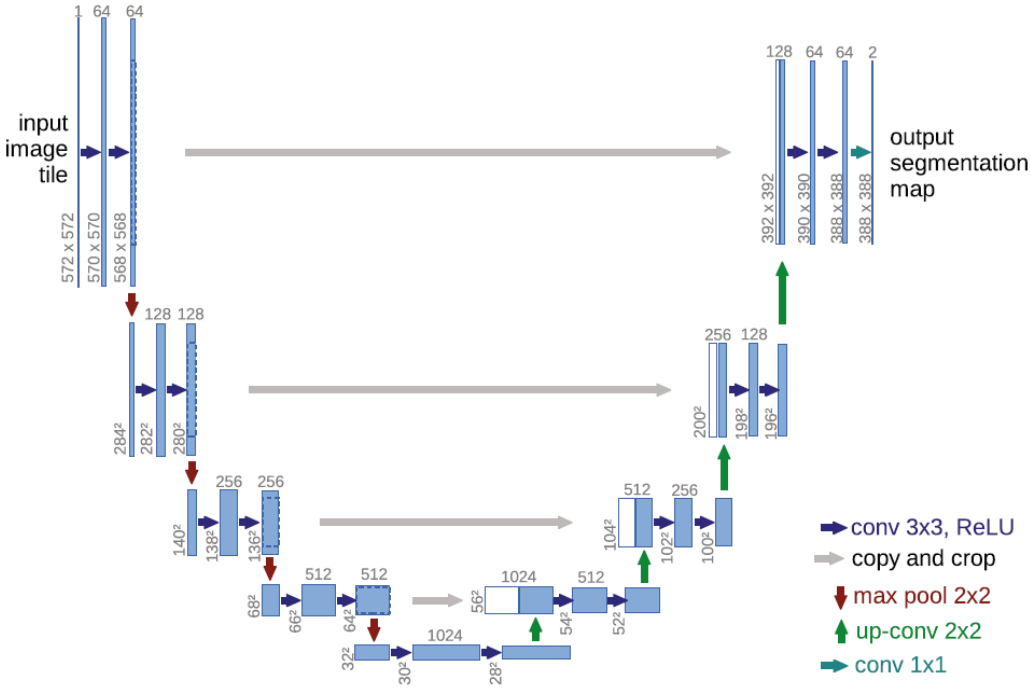
2.3.1 Arsitektur U-Net

U-Net menjadi salah satu teknik segmentasi yang sedang berkembang berdasarkan arsitektur *autroencoder*. U-Net memiliki tugas yang dapat memberikan kelas atau wilayah lokal di sekitar piksel pada hasil segmentasi citra. U-Net memiliki kerangka kerja yang terdiri dari *contracting path* dan *expansive path* dan berbentuk huruf U (Ronneberger dkk., 2015).

Contracting path terletak pada sisi kiri yang dibentuk dari jaringan konvolusi berulang serta diikuti dengan operasi fungsi aktivasi ReLU dan *max pooling*.

[1] https://www.mathworks.com/content/mathworks/www/en/discovery/convolutional-neural-network/jcr:content/mainParsys/image_copy.adapt.full.high.jpg/1508999490138.jpg

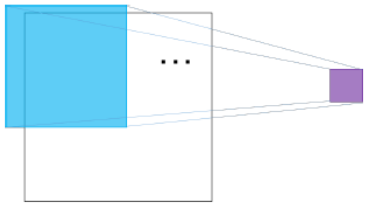
Informasi spasial yang disimpan akan berkurang pada lapisan sebelumnya namun informasi fitur yang dihasilkan dari pemrosesan semakin ditingkatkan. *Expansive path* terletak pada sisi kanan yang dibentuk dari jaringan konvolusi berulang yang menggabungkan fitur dengan informasi spasial melalui urutan *up-sampling* dan penggabungan lapisan resolusi tinggi dari jalur *contracting path* dengan *output up-sampling* yang disebut sebagai *skip-connection. Skip-connection* menjadi solusi dari kekurangan *autoencoder* dimana resolusi saat proses *encoding* hilang sehingga berdampak pada hasil citra yang buram. Berikut adalah contoh arsitektur U-Net (Gambar 2.7).



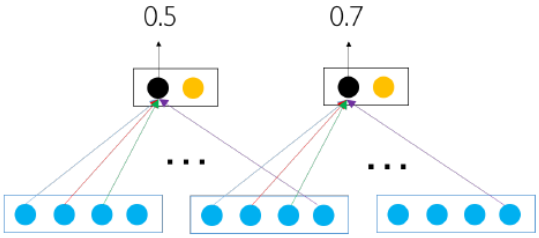
Gambar 2.7 Arsitektur U-Net (Ronneberger dkk., 2015).

2.3.2 *Convolution Layer*

Convolution layer bekerja dengan prinsip *sliding window* (Gambar 2.8) dan *weight sharing* (Gambar 2.9). Tahap ini mentransformasi suatu *window* menjadi suatu nilai numerik *(filter). Window* ini kemudian digeser-geser sebanyak T kali, sehingga akhirnya didapatkan vektor dengan panjang *d* x T (Putra, 2020).



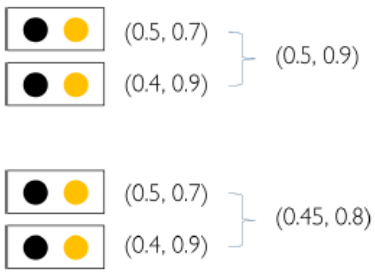
Gambar 2.8 *Sliding window.*



Gambar 2.9 *Weight sharing.*

2.3.3 *Pooling Layer*

Tahap ini mengkombinasikan atau meringkas semua vektor yang dihasilkan dari *convolution layer* menjadi satu vektor c. Adapun teknik *pooling* yang sering digunakan, diantaranya: *max pooling* dan *average pooling. Max pooling* mencari nilai maksimum untuk setiap dimensi vektor. *Average pooling* mencari nilai rata-rata setiap dimensi vektor (Paniagua dan Bedmar, 2018). Berikut paparan ilustrasi mengenai *max pooling* dan *average pooling* (Gambar 2.10).



Gambar 2.10 Contoh *max pooling* (atas) dan *average pooling* (bawah).

2.3.4 *Fully-Connected Layer*

Setelah tahap *pooling* selesai maka *multidimensional array* tersebut akan dilewatkan pada *multilayer perceptron (fully-connected)* untuk mendapatkan hasil prediksi baik berupa klasifikasi gambar, dsb. Pada tahap *fully-connected layer*,

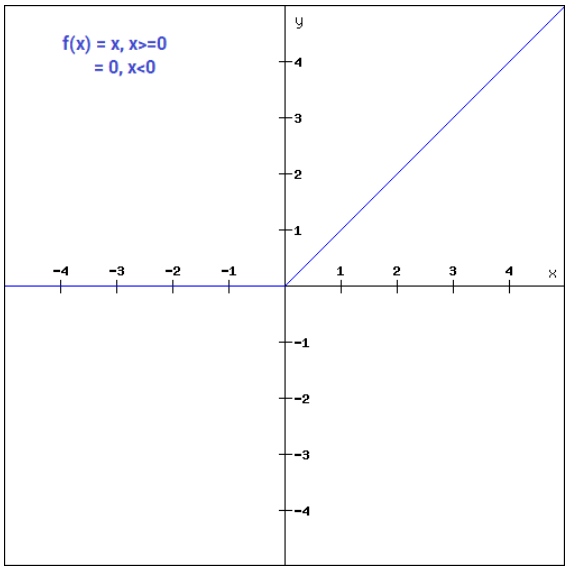
semua *neuron* aktivasi dari lapisan sebelumnya terhubung dengan *neuron* aktivasi dari lapisan selanjutnya dengan cara *flatten* (*reshape feature map*) menjadi sebuah vektor sehingga dapat digunakan sebagai *input* dari *fully-connected layer* (Yamashita dkk., 2018).

2.3.5 Fungsi Aktivasi

Fungsi aktivasi adalah suatu fungsi yang digunakan untuk mengaktifkan atau tidak mengaktifkan neuron pada jaringan syaraf buatan. Ada beberapa fungsi aktivasi yang sering digunakan antara lain sebagai berikut:

1. ReLU (*Rectified Linear Unit*)

ReLU merupakan salah satu fungsi aktivasi yang cukup populer. Fungsi ini membuat pembatas pada bilangan nol, yang artinya apabila x $\leq$ 0 maka x = 0 dan apabila x > 0 maka x adalah nilai x itu sendiri (LeCun dkk., 2015). Grafik fungsi aktivasi ReLU dapat diilustrasikan seperti pada (Gambar 2.11).



Gambar 2.11 Fungsi aktivasi ReLU (Anonim, 2020).

Adapun persamaan matematis dari fungsi aktivasi ReLU ditunjukkan pada persamaan 2.7.

$$f(x) = \max(0, x) \tag{2.7}$$

ReLU sangat mempercepat proses konvergensi yang dilakukan dengan *stochastic gradient descent* jika dibandingkan dengan fungsi aktivasi *sigmoid* ataupun *tanh*. Namun jika inisialisasi *learning rate* pada model terlalu tinggi maka unit ReLU dapat menjadi rapuh dan tidak berfungsi (Pramesti Hatta K, 2017).

2. *Softmax*

Fungsi *softmax* menghitung probabilitas terhadap sejumlah kejadian. Fungsi aktivasi ini mentransformasi c agar jumlah semua nilainya berada pada rentang 0 sampai 1. Adapun persamaan matematis dari fungsi *softmax* ditunjukkan pada persamaan 2.8 (Agarap, 2019).

$$c_i = \frac{e^{(x.W+b)_{[i]}}}{\sum_j e^{(x.W+b)_{[j]}}} \tag{2.8}$$

Dimana c_i adalah elemen vektor ke-i. *Output* yang berupa distribusi probabilitas tersebut dapat digunakan untuk menghitung *loss* dengan konsep *cross entropy.*

2.3.6 *Cross Entropy Loss Function*

Loss merupakan ukuran seberapa dekat atau berbeda model yang dihasilkan dengan konsep asli. Secara umum terdapat dua jenis *loss*, yaitu *generalization loss* dan *training loss. Generalization loss* adalah ukuran sejauh mana algoritma mampu memprediksi *unobserved data* dengan tepat, hal ini dikarenakan terbatasnya data dalam membangun sebuah model sehingga dapat menimbulkan ketidakcocokan dengan data yang asli (Yamashita dkk., 2018). Untuk mengukur nilai *loss* dapat diekspresikan dengan *loss function.* Salah satu fungsi yang dapat menghitung *loss* yaitu *cross entropy loss*, yang dapat dilihat pada persamaan 2.9.

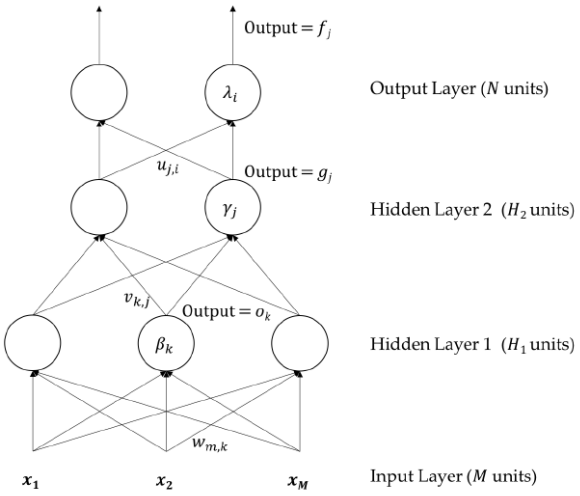
$$H = -\sum_{i=1}^{N} q(x)\log\left(p(x)\right) \tag{2.9}$$

Dimana *q(x)* adalah distribusi data asli dan *p(x)* adalah sampel data dengan distribusi.

Nilai *entropy* yang tinggi melambangkan distribusi lebih *uniform* sedangkan kemampuan klasifikasi model yang baik yaitu dengan nilai *entropy* rendah yang melambangkan distribusi tersebut tidak *uniform* (Putra, 2020).

2.3.7 *Feed Forward* dan *Backpropagation*

Secara umum *deep neural network* terdapat tiga jenis *layers* yaitu *input layer, hidden layer,* dan *output layer* yang diilustrasikan pada (Gambar 2.12).



Gambar 2.12 Proses *feed forward* dan *backpropagation deep neural network*.

Input layer yang merupakan nilai masukan yang tidak dilakukan operasi apapun, kemudian nilai *input* diberikan ke *hidden units.* Pada *hidden units* ini, nilai *input* diproses lalu dihitung hasil fungsi aktivasi untuk setiap neuron, kemudian hasil tersebut diberikan ke *layer* berikutnya. Dilakukan terus menerus hingga *layer* terakhir yaitu *output layer.* Proses ini dinamakan *feed forward.* Untuk mendapatkan hasil *output* yang bagus maka kita perlu memperbaharui parameter *weights* secara bertahap dari *output* ke *input layer* berdasarkan *error/loss.* Proses ini dinamakan *backpropagation.*

Secara matematis, proses *feed forward* dan *backpropagation* dapat dilihat pada persamaan 2.10 – 2.15 berdasarkan *neural network* pada (Gambar 2.12).

(1) *Input layer* ke *hidden layer* 1

$$o_k = \sigma\left(\sum_{m=1}^{M} x_m w_{m,k} + \beta_k\right) \tag{2.10}$$

(2) *Hidden layer* 1 ke *hidden layer* 2

$$g_j = \sigma\left(\sum_{k=1}^{H_1} o_k v_{k,j} + \gamma_j\right) \tag{2.11}$$

(3) *Hidden Layer* 2 *to Output*

$$f_i = \sigma\left(\sum_{m=1}^{H_2} g_j u_{j,i} + \lambda_i\right) \tag{2.12}$$

(4) *Output* ke *hidden layer* 2

$$\begin{aligned} \delta_i &= (y_i - f_i) f_i (1 - f_i) \\ \Delta u_{j,i} &= -\eta(t)\delta_i g_j \\ \Delta\lambda_i &= -\eta(t)\delta_i \end{aligned} \tag{2.13}$$

(5) *Hidden layer* 2 ke *hidden layer* 1

$$\begin{aligned} \varphi_j &= \sum_{i=1}^{N} \delta_i u_{j,i} g_j \left(1 - g_j\right) \\ \Delta v_{k,j} &= -\eta(t)\varphi_j o_k \\ \Delta\gamma_j &= -\eta(t)\varphi_j \end{aligned} \tag{2.14}$$

(6) *Hidden layer* 1 ke *input layer*

$$\begin{aligned} \mu_k &= \sum_{j=1}^{H_2} \varphi_j v_{k,j} o_k \left(1 - o_k\right) \\ \Delta w_{m,k} &= -\eta(t)\mu_k x_m \\ \Delta\beta_k &= -\eta(t)\beta_k \end{aligned} \tag{2.15}$$

β, γ, λ adalah *noise* atau *bias*, σ adalah fungsi aktivasi, sedangkan u, v, w adalah bobot (Rumelhart dkk., 1986).