# LANDASAN TEORI

Pada bab ini membahas seluruh teori yang dipakai oleh penulis dalam melaksanakan proses penelitian.

## 2.1 *Computer Vision*

Visi komputer adalah cabang ilmu komputasi sains yang memelajari seperti apakah komputer dapat memahami gambar dan video secara mendalam.

“Dari perspektif ilmu teknis, tujuan utama dari visi komputer adalah untuk mengembangkan sistem yang dapat melakukan tugas-tugas yang mirip dengan sistem visual makhluk hidup.” (Huang 1996, 1).

Komputer diharapkan untuk dapat memeriksa citra digital yang kompleks secara memadai, sehingga data yang diperoleh dari citra digital dapat dianalisis untuk memperoleh informasi yang dapat diproses lebih lanjut. *Computer vision* sendiri memiliki banyak aplikasi seperti membantu manusia dalam mengidentifikasi tugas-tugas tertentu, mengendalikan proses, mengenali peristiwa, interaksi manusia dan komputer.

## 2.2 *Deep Learning*

*Deep Learning* merupakan suatu metode pembelajaran mesin yang memfokuskan pada pemanfaatan banyaknya layer untuk melakukan pemrosesan informasi berupa ekstraksi serta transformasi atau perubahan data, hal ini guna untuk menganalisis dan mengklasifikasikan pola, terutama untuk mempelajari data seperti gambar, audio, dan teks. Pembelajaran mendalam juga dikenal sebagai jaringan saraf dalam. “Fokus dari *Deep Learning* adalah bagaimana menggambarkan kesimpulan dari data mentah tanpa adanya bantuan dari dalam diantara dua feature engineering” (Gad 2018, xix). Pembelajaran mendalam itu sendiri berada di persimpangan beberapa bidang Pembelajaran Jaringan Syaraf Tiruan (JST).

Penyebab popularitas pembelajaran mendalam saat ini adalah peningkatan kekuatan komputer dalam melakukan komputasi, peningkatan jumlah data yang dalam proses pelatihan, serta meningkatkan riset pembelajaran mesin.

## 2.3 Memahami *Inception Model*

*Inception Model* adalah terobosan di era *deep learning* yang disebut *Deep Convolutional Neural Network*, yang dianggap sebagai klasifikasi canggih dan model pengenalan visual di *ImageNet*. Ide utama di balik *indeption* *model* adalah untuk meningkatkan pemanfaatan sumber daya komputasi di dalam jaringan saraf tiruan. Model ini digunakan untuk meningkatkan kedalaman fitur dengan menjaga biaya komputasi tetap konstan. Selain itu, ia menyediakan cabang komputasi paralel dari lapisan konvolusi untuk masukan yang sama dan menggabungkan keluaran dari semua lapisan paralel sebelum meneruskan ke lapisan berikutnya dalam arsitektur.

Model awal dasar terdiri dari 4 lapisan paralel. Lapisan paralel pertama memiliki konvolusi 1x1. Lapisan paralel kedua memiliki konvolusi 1x1 diikuti oleh konvolusi 3x3. Lapisan paralel ketiga memiliki konvolusi 1x1 diikuti oleh konvolusi 5x5. Keempat lapisan paralel memiliki konvolusi 1x1 diikuti oleh lapisan maxpool. Lapisan terakhir menggabungkan semua keluaran dari lapisan paralel, sebelum memasukkannya ke lapisan berikutnya.

## 2.4 Model Pra-Terlatih

Model terlatih (*Pretrained Model*) berisi bobot yang sudah dilatih untuk jaringan neural tertentu. Salah satu model terlatih adalah *ImageNet*, yang telah melatih lebih dari 1,3 juta gambar untuk tugas deteksi dan pengenalan objek. Biasanya peneliti menghapus lapisan klasifikasi akhir atau lapisan yang sepenuhnya terhubung dari model yang telah dilatih sebelumnya dan menggantinya dengan lapisan klasifikasi SVM atau KNN untuk penelitian mereka.

(Jeff Donahue 2013) menyelidiki hal itu untuk menggeneralisasi model ImageNet yang selanjutnya dapat digunakan untuk dataset lain pada berbagai kedalaman. Mereka melakukan ini dengan memvisualisasikan pemisahan antara kategori yang berbeda di lapisan pertama dan keenam, menunjukkan pemisahan yang lebih besar di lapisan yang lebih dalam. Mereka telah menunjukkan bahwa jaringan delapan lapis yang memiliki tiga lapisan yang terkoneksi penuh adalah yang paling mahal dalam hal waktu komputasi. Namun, dengan mengatur jaringan yang telah dilatih sebelumnya secara terpisah untuk klasifikasi *soft bird*, adaptasi domain, atau pelabelan *view*, itu mengungguli model mutakhir dalam kategori ini.

Selain itu, *FaceNet* telah meraih kesuksesan dalam pengenalan wajah dengan mengekstraksi fitur *embedding* dari model awal dan memiliki pengklasifikasi SVM pada lapisan akhir. Dalam salah satu eksperimen kami, model terlatih FaceNet telah digunakan untuk mengekstrak fitur embedding dari kumpulan data kami untuk melatih model SVM di lapisan klasifikasi akhir.

Umumnya, model terlatih tersedia dalam satu file protobuf dengan file meta, checkpoint, dan grafik. Berikut adalah file yang ada dalam model terlatih.

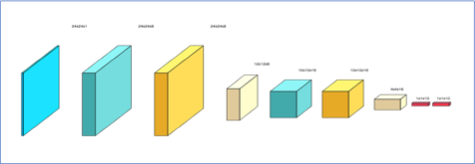
1. model.meta
2. model.Indeks
3. pos pemeriksaan
4. model data
5. model.pbtxt

Langkah-langkah berikut harus dilakukan, saat memuat model ke sesi tensorflow sebelum memulai proses pelatihan.

1. Dasar komputasi di *TensorFlow* adalah objek Grafik yang dimuat pertama kali ke sesi tensorflow atau pelatihan jaringan neural dalam apa pun.
2. Sesi default *Tensorflow* menyimpan jaringan node, bobot terlatihnya yang terkait, node operasional seperti softmax, penambahan, perkalian, dan ini terhubung satu sama lain.
3. Objek grafik dibuat, dapat diakses melalui *as\_graph\_def ()*, yang mengembalikan objek *GraphDef* dari sesi *tensorflow*.
4. Selain itu, *placeholder* input, operasi, dan variabel dapat diakses dari objek grafik *tensorflow* yang ada dalam sesi *tensorflow*.
5. Objek grafik ini digunakan untuk menjalankan model.

## 2.6 *Convolutional Neural Network*

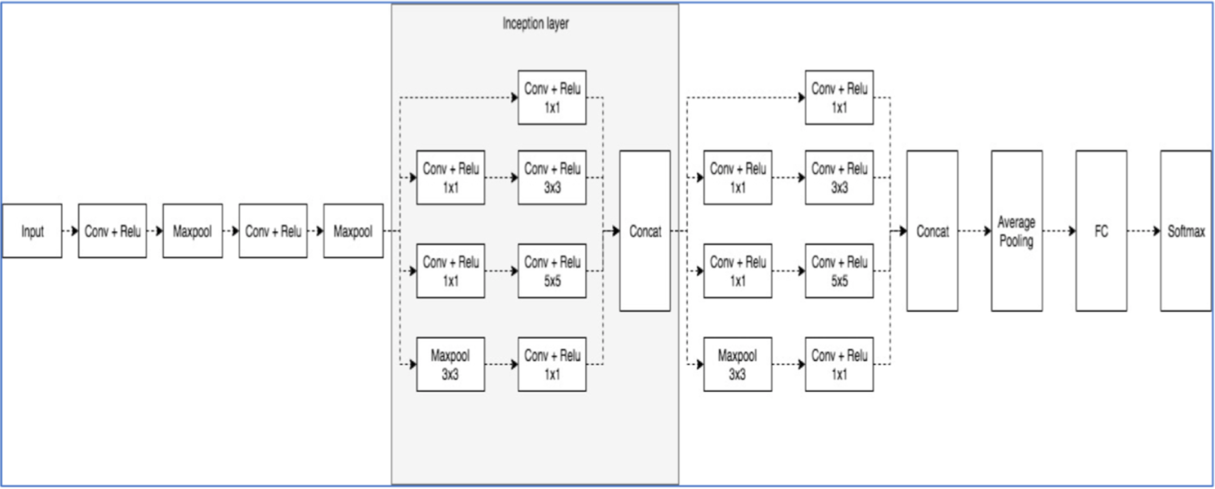
Model tersebut diperkenalkan Mahasiswa *Hardvard University* bernama Yann LeCunn pada tahun 1998 untuk *Optical Character Recognition* (OCR), di mana mereka telah menunjukkan kinerja yang mengesankan pada pengenalan karakter. Selain itu, CNN juga digunakan pada tugas yang bersinggungan dengan gambar, serta digunakan untuk pengenalan sinyal dan bahasa, spektogram *audio, video*, dan gambar volumetrik. Gambar 2 menunjukkan diagram blok tingkat tinggi CNN.



Gambar 2.1 Diagram blok arsitektur Convolutional Neural Network

adalah lapisan masukan, diikuti oleh blok warna biru yang merupakan lapisan konvolusi. Blok warna kuning adalah lapisan aktivasi, dan blok warna putih adalah lapisan *maxpool*. Terakhir, blok berwarna merah adalah lapisan yang sepenuhnya terhubung dalam arsitektur CNN. Arsitektur di atas adalah model dasar untuk pemahaman awal tentang CNN.

Ini adalah terobosan dalam area pembelajaran yang mendalam ketika para peneliti merancang model *Inception* pada tahun 2014. Menurut (Christian Szegedy 2014)“Lebih dalam dengan konvolusi”, lapisan awal diperkenalkan ke model CNN yang ada, yang telah menetapkan tolok ukur dalam klasifikasi canggih dan tugas deteksi pada *dataset ImageNet*. Fungsionalitas utama dari lapisan awal ini adalah untuk meningkatkan pemanfaatan sumber daya komputasi di dalam jaringan saraf. Gambar 2.2, menunjukkan contoh bentuk umum diagram blok dari *inception layer*.



Gambar 2.2 Bentuk umum diagram blok dari inception layer.

Dalam diagram di atas, blok warna abu-abu menunjukkan lapisan permulaan dasar. Satu-satunya perbedaan antara model CNN dan *Inception* adalah lapisan awal, yang memiliki 4 konvolusi terhubung paralel dan lapisan kumpulan maksimum dengan lapisan gabungan di bagian akhir.

Sejauh ini, kami telah mendapatkan gambaran umum tentang arsitektur model CNN dan *Inception* dan apa arti dari model *deep learning*. Terminologi umum yang digunakan dalam arsitektur *deep learning* telah dijelaskan di bagian berikut.

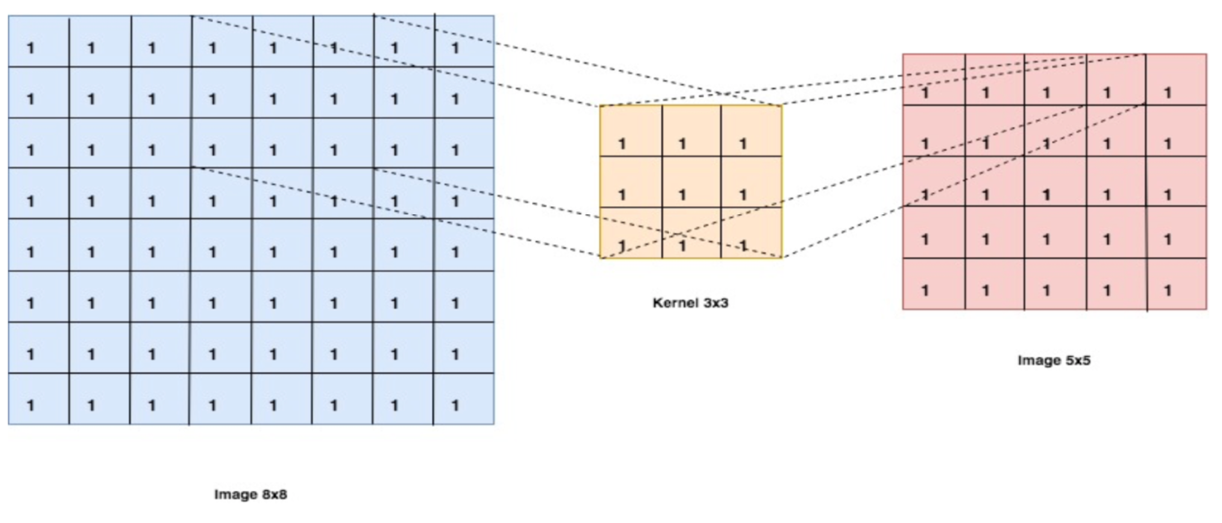
## 2.6.1 Lapisan Konvolusional

Lapisan Konvolusional atau sering disebut dengan *Convolution Layer* menyediakan operasi konvolusi, di mana filter 2-D atau 3-D dengan ukuran yang sesuai menyapu gambar dan menerapkan filter ke setiap kedalaman gambar. Lapisan konvolusional adalah versi terbatas dari *Multi-Layer Perceptron* (MLP) yang diadaptasi untuk mengambil input 2D / 3D, bukan 1D. Ide di balik lapisan konvolusional adalah untuk mendeteksi fitur dasar seperti tepi, sudut, dan titik akhir, dan menggabungkannya menggunakan beberapa lapisan untuk mendapatkan fitur tingkat tinggi yang mungkin mendeskripsikan objek secara lengkap. Gambar 4, menunjukkan konvolusi sampel gambar pada ukuran filter 3x3.

Operasi ini dilakukan dengan cara berikut: lapisan pertama f1 (·) berisi fitur paling dasar, lapisan kedua f2 (f1 (·)) adalah fungsi fitur dasar di lapisan sebelumnya. Lapisan ketiga kemudian merupakan fungsi f3 (f2 (f1 (·))) dari fitur-fitur di lapisan kedua dan seterusnya.

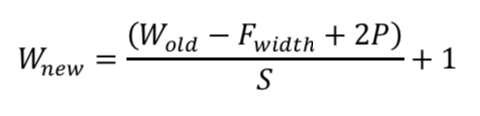
Selain itu, arsitektur ini dirancang untuk ekstraksi fitur tingkat tinggi dari gambar pada lapisan tertentu untuk mendeskripsikan objek seperti wajah, kursi, atau mobil. Selain itu, konvolusi juga menyediakan atribut fitur penting dan berharga yang disebut invariansi pergeseran. Artinya, jika masukan ke lapisan pertama digeser, maka keluaran lapisan pertama juga bergeser dengan jumlah yang sama. Konvolusi memiliki 2 parameter utama yang dapat mengubah perilaku konvolusi, seperti langkah dan bantalan. *Padding* dan *Stride* dijelaskan di bagian selanjutnya.

Gambar berikut menunjukkan konvolusi pada ukuran gambar 8x8 dengan Stride 1 dan Padding 0 (sama) dan ukuran kernel 3x3.



Gambar 2.3 Contoh penerapan Jaringan Convolution pada gambar gray scale pada ukuran kernel 3x3.

Keluaran dari lapisan konvolusi dihitung sesuai rumus berikut.



Fwidth: Filter atau ukuran kernel seperti parameter lebar dan tinggi saat menggunakan rumus masing-masing.

P: Padding

S: Ukuran *stride window* untuk konvolusi

Wnew: Lebar baru dari gambar keluaran

Wold: Lebar lama gambar masukan

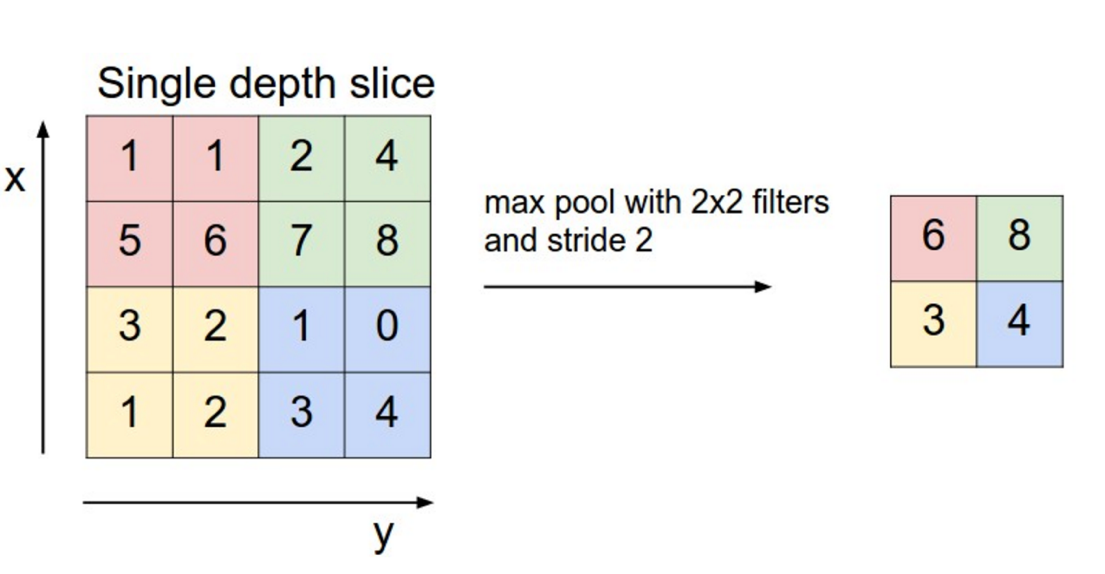
## 2.6.2 *Pooling Layer*

*Pooling* adalah metode untuk mengurangi ukuran fitur dalam lebar dan tinggi suatu input. Operasi penggabungan menyapu jendela persegi panjang di atas fitur masukan dan menghitung operasi pengurangan ukuran untuk setiap jendela (rata-rata, maksimum, atau nilai maksimum dengan *argmax*). Setiap operasi penggabungan menggunakan jendela persegi panjang ukuran k, dipisahkan oleh langkah *offset*. Misalnya, jika langkah semuanya adalah setiap jendela digunakan, jika langkah semua dua setiap jendela alternatif digunakan di setiap dimensi. cara sederhana untuk mengurangi ketepatan posisi dari lokasi fitur-fitur khusus di peta fitur. Karena posisi pasti dari fitur tersebut tidak relevan, yang penting hanya posisinya dalam kaitannya dengan fitur lain, terutama untuk tugas klasifikasi.

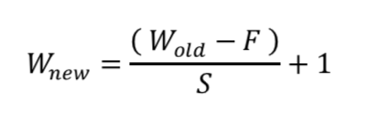
Dengan kata lain, kami tidak peduli di mana tepi atau sudut terletak pada gambar, kami hanya peduli tentang posisinya relatif terhadap sudut dan tepi lain dalam gambar. Besar kecilnya *field* reseptif yang kritis, yang paling sering digunakan adalah 2x2, hal ini disebabkan hilangnya informasi yang signifikan yang terjadi saat menggunakan *field* reseptif yang lebih besar. Misalnya, jika bidang reseptif 2x2 digunakan, empat piksel diubah menjadi satu. Meningkatkan bidang reseptif ke 3 × 3 berarti sembilan piksel diubah menjadi satu dan jika 4 × 4 digunakan, 16 piksel diubah menjadi satu dan seterusnya.

Sebagai gambaran, bayangkan Anda memiliki gambar 12 × 12, menghasilkan total 144 piksel. Bidang reseptif 2 × 2 akan memberi Anda gambar 6 × 6 dengan 36 piksel. Anda sekarang telah kehilangan 75% informasi, tetapi objek tersebut mungkin masih dapat dikenali. Jika sebaliknya, Anda menggunakan 3 × 3 Anda hanya mendapatkan gambar 4 × 4 dengan 16 piksel menyisakan hanya 11% dari informasi asli.

Sekarang mungkin untuk mengenali bentuk aslinya, tetapi jika Anda tidak beruntung Anda kehilangan terlalu banyak informasi. Jika Anda memutuskan untuk mendorongnya lebih jauh dengan menggunakan penggabungan 4 × 4 akan memberi Anda gambar 3 × 3 dengan total sembilan piksel, Anda sekarang telah kehilangan 94% dari semua informasi dalam gambar. Gambar 5, menunjukkan operasi kumpulan maksimal pada filter 2x2 dengan langkah 2x2.



Gambar 2.4 Operasi pada Layer Maxpool.



W*new*: Lebar baru untuk gambar keluaran

W*old*: lebar dari gambar masukan

F: Ukuran *Filter Width*

S: Ukuran *stride*

## 2.6.3 Lapisan Terhubung Sepenuhnya

Lapisan Terhubung Sepenuhnya (*Fully Connected Layer*) atau ANN di mana *neuron* dalam satu *layer* saling berinteraksi ke dengan neuron pada *layer* setelahnya. Prinsip ini sama dengan model jaringan saraf *multi-layer* *perceptron* tradisional dan cara kerjanpun juga sama. Lapisan yang terhubung sepenuhnya mengacu pada lapisan terakhir dalam model CNN lengkap. Lapisan yang sepenuhnya terhubung beroperasi sebagai *Multi-Layer Perceptron* (MLP) dengan biasanya dua atau tiga lapisan tersembunyi dan satu lapisan klasifikasi. Sifat-sifat MLP membuatnya menjadi perkiraan fungsi yang luar biasa, dengan hanya dua lapisan tersembunyi ia dapat mendekati fungsi apa pun dengan asumsi ia memiliki cukup *neuron* tersembunyi. Biasanya, jumlah neuron di lapisan tersembunyi adalah konstan, dengan 4096 adalah jumlah umum untuk jaringan dalam dengan gambar masukan besar.

Input ke lapisan tersembunyi pertama berasal dari semua neuron di lapisan sebelumnya (baik lapisan penggabungan atau konvolusional). Dengan kata lain, neuron pada *layer* sebelumnya terhubung dengan neuron di *hidden* *layer* pertama. Keluaran dari lapisan tersembunyi pertama terhubung ke setiap neuron di lapisan tersembunyi kedua, itu sepenuhnya terhubung. Keluaran dari lapisan tersembunyi terakhir kemudian sepenuhnya terhubung ke lapisan klasifikasi akhir. Ukuran lapisan terakhir bergantung pada jumlah kelas yang digunakan untuk melatih jaringan saraf.

Lapisan yang sepenuhnya terhubung dapat diratakan dan dihubungkan ke lapisan keluaran dan seterusnya ukurannya diperkecil untuk klasifikasi gambar. Pada lapisan yang sepenuhnya terhubung, jika masukan berasal dari lapisan konvolusi atau lapisan *maxpool* berukuran XxYxZ, kita dapat memilih berapa banyak *node* yang kita butuhkan dalam lapisan pipih yang terhubung sepenuhnya. Bisa berupa jumlah node XxYxZ atau (X \* Y \* Z) / 2 jumlah node dan kemudian output akan dikurangi *feed* ke lapisan klasifikasi output.

## 2.6.4 Fungsi Aktivasi

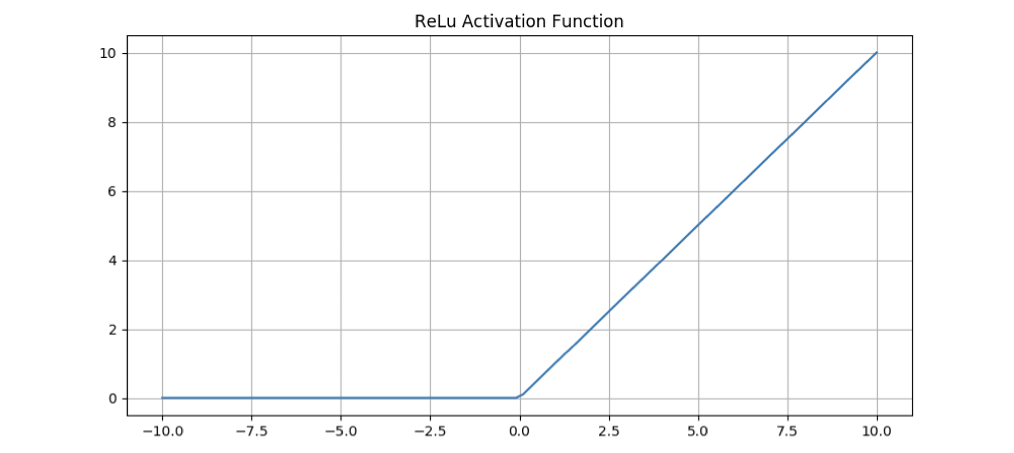
Fungsi aktivasi sangat penting untuk jaringan saraf dalam, yang rumit dan kompleks. Mereka membawa properti non-linearitas ke jaringan saraf. Properti utama dari fungsi aktivasi adalah mengubah sinyal input menjadi sinyal output. Ini digunakan di setiap *node* dari *deep neural network* untuk representasi abstraksi dari potensi aksi yang menembakkan *node*.

Jika kita tidak menggunakan fungsi aktivasi, fungsi pemetaan keluaran akan, secara default merupakan fungsi linier, yang linieritasnya kurang efektif untuk mempelajari batas fungsi kompleks dari data masukan. Berikut adalah beberapa fungsi aktivasi yang dijelaskan secara rinci.

### 2.6.4.1 *ReLU*

Fungsi aktivasi memainkan peran sentral dan sangat penting dalam cara kerja CNN dan seberapa baik kinerjanya. Setiap *neuron* di setiap lapisan konvolusional dan terhubung sepenuhnya memiliki fungsi aktivasi non-linier tertentu. Jenis fungsi non-linier bervariasi, tetapi sebagian besar jaringan saat ini menggunakan ReLU (*Reflective Linear Unit*) yang secara matematis dijelaskan oleh φ (x) = *max*(0, x).

Pada beberapa waktu terakhir, ReLU lebih sering digunakan dari pada Tanh karena kemampuan konvergensinya melampaui fungsi tanh sebanyak 6 kali. Ini mengatasi dan memperbaiki masalah gradien yang hilang (*vanishing gradient*).



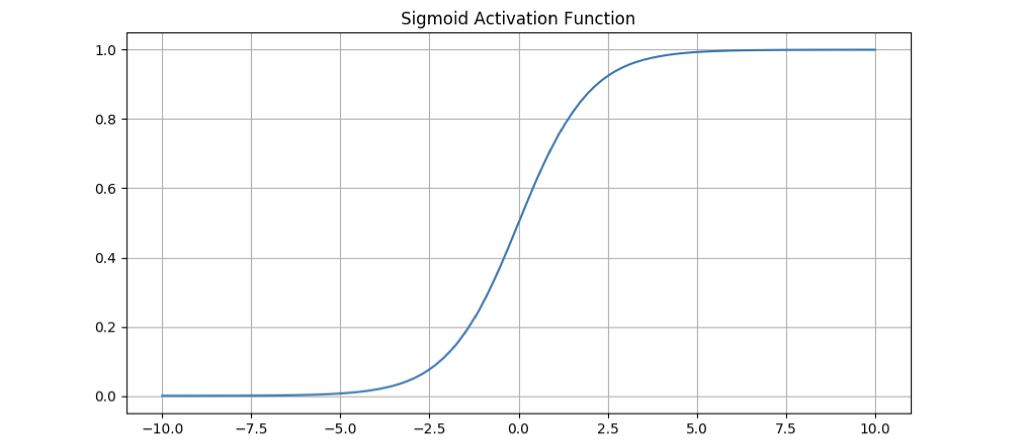
Gambar 2.5 Fungsi aktivasi ReLU.

### 2.6.4.2 *Sigmoid*

*Fungsi sigmoid* digunakan sebagai fungsi aktivasi adalah kasus khusus dari fungsi logistik yang ditunjukkan pada 7 pertama ini termasuk fungsi tangen logistik dan hiperbolik. Kurva *sigmoid* juga umum dalam statistik sebagai fungsi distribusi kumulatif (dari 0 hingga 1).

Berikut alasan utama mengapa fungsi ini kurang populer,

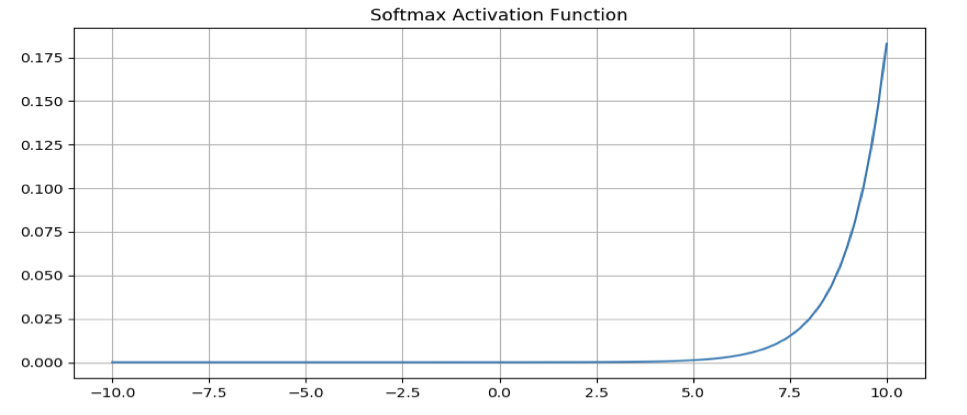
1. Masalah *Vanishing Gradient*
2. Output tidak berada di tengah nol
3. *Sigmoid* menjenuhkan dan membunuh gradien
4. *Sigmoid* memiliki konvergensi yang lambat



Gambar 2.6 Fungsi aktivasi Sigmoid.

### 2.6.4.3 *Softmax*

Fungsi *softmax* ini juga dikenal sebagai fungsi eksponensial ternormalisasi yang digunakan sebagai fungsi aktivasi di jaringan saraf tiruan. Ini digunakan sebagai distribusi probabilitas untuk kemungkinan hasil-K. Ini juga digunakan di lapisan akhir dengan *cross entropy* di jaringan saraf untuk distribusi probabilitas untuk data kategori. *Softmax* memberikan keluaran dengan rentang antara 0-1.

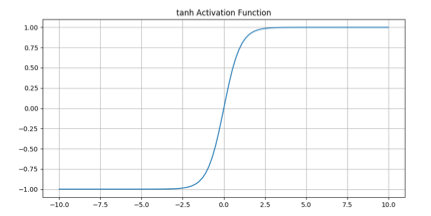


Gambar 2.7 Fungsi aktivasi Softmax.

### 2.6.4.4 *TanH*

Fungsi tanh adalah fungsi aktivasi lain dari jenis *Logistic sigmoid*, yang memiliki kurva berbentuk "s", tetapi nilai keluaran berkisar dari -1 hingga 1. Fungsi ini membuat masukan negatif memetakan secara kuat ke keluaran negatif. Selain itu, hanya input bernilai nol yang dipetakan ke output bukan nol terdekat. Properti ini membuat jaringan lebih kecil kemungkinannya untuk bottleneck selama pelatihan dan menghilangkan kemungkinan hilangnya gradien (*vanishing gradient*). Menghitung gradien untuk fungsi *Tanh* juga menggunakan aturan hasil bagi.

Keluarannya berada di tengah nol karena berada di antara -1 hingga 1. Pengoptimalan lebih mudah daripada sigmoid tetapi tetap saja, ia memiliki masalah gradien menghilang (*vanishing gradient*).



Gambar 2.8 Fungsi aktivasi Tanh

## 2.6.5 Fitur Dasar

Fitur dasar (*elementary feature*) adalah bentuk yang paling sederhana, seperti tepi dan sudut dalam gambar yang dideteksi oleh beberapa detektor fitur. *Operator Sobel* adalah contoh paling sederhana dari pendeteksi fitur. Ini mendekati turunan dari nilai piksel. Jenis detektor ini adalah dasar dari CNN, ada banyak detektor fitur yang bertanggung jawab untuk menemukan fitur dasar di setiap lapisan jaringan neural. Fitur dasar di satu bagian gambar digunakan di bagian lain gambar dan karena satu fitur ini peta memiliki *Weight Vector* yang identik meskipun bidang reseptif lokalnya berada di wilayah gambar yang berbeda. Setiap unit seperti pada neuron di setiap peta fitur memiliki vektor bobot yang sama persis. Dan konsep ini membantu dalam hal mengikuti secara detail.

1. Kemampuan generalisasi
2. Pengurangan waktu pelatihan
3. Hal ini memungkinkan adanya paralelisme

## 2.6.6 Vektor Bobot (*Weigth Vector*)

Vektor bobot adalah filter di *Convolutional Neural Network* dan membantu dalam proses konvolusi serta mengetahui peta fitur melalui operasi ekstraksi fitur. Awalnya, matriks bobot dibuat oleh distribusi bilangan *Gaussian* yang memiliki mean 0 dan standar deviasi 0,001. Inisialisasi ini juga dapat dicapai dengan menyetel semua bobot ke angka acak atau nol di semua lapisan jaringan saraf. Inisialisasi acak ini memengaruhi konvergensi fungsi kerugian, yang akan membutuhkan banyak waktu untuk mencapai minimum lokal atau minimum global.

Selain itu, bobot dibagi di antara peta fitur. Jika dua neuron memiliki peta fitur yang sama, maka mereka berbagi vektor bobot yang sama untuk mengurangi ruang dan kompleksitas komputasi. Umumnya peneliti menggunakan vektor bobot seragam acak untuk ciri distribusi normalnya.

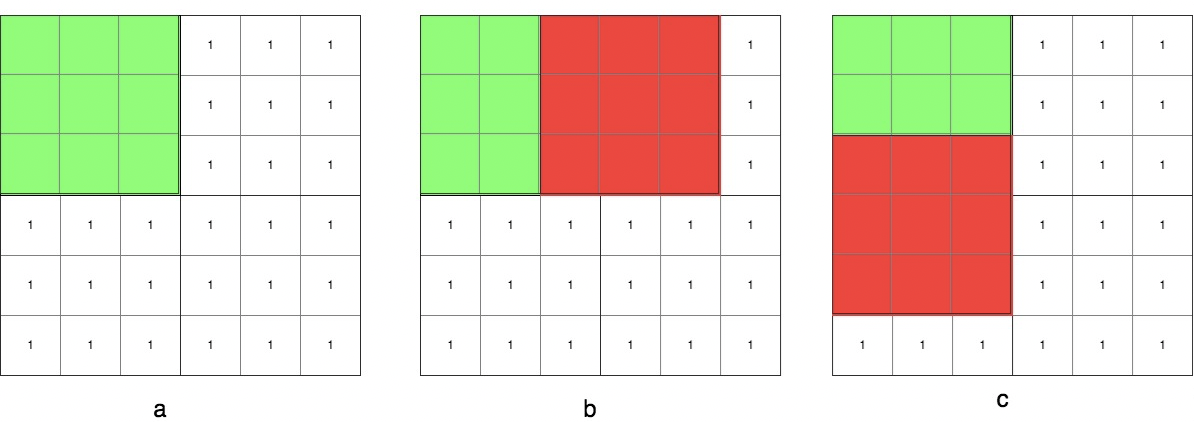
## 2.6.7 *Filter* atau *Kernel*

*Kernel* dan *filter* digunakan secara bergantian dalam laporan ini. *Kernel* adalah matriks 2-D atau 3-D, yang digunakan dalam proses konvolusi seperti yang dijelaskan sebelumnya di bagian ini. Setiap konvolusi memunculkan sejumlah saluran seperti yang ditentukan oleh jumlah *filter*. *Kernel* umumnya disebut sebagai detektor fitur yang digunakan dalam lapisan konvolusi.

Kernel didefinisikan sebagai 3x3, 5x5, 7x7, dan seterusnya. Ini berarti tinggi dan lebar *kernel*. *Filter* atau *kernel* yang lebih kecil menangkap fitur detail menit dalam gambar daripada kernel yang lebih besar yang meninggalkan beberapa fitur penting dalam konvolusi atau di lapisan kumpulan maksimal. Jadi, peneliti lebih memilih menggunakan kernel yang lebih kecil daripada kernel yang lebih besar seperti 11x11 atau 19x19. *Kernel* didefinisikan dalam *Tensorflow* sebagai format vektor [1,3,3,1], ini berarti, (*batch, height, width, depth*) masing-masing. Tinggi dan lebar ditentukan untuk ukuran kernel 3x3 atau 5x5. Selain itu, ukuran tumpukan ditentukan sesuai kebutuhan pelatihan dan kedalaman ditentukan oleh kedalaman gambar masukan.

## 2.6.8 *Strides*

*Strides* adalah sebuah konsep yang mengontrol pergerakan kernel di atas gambar dalam operasi konvolusi. Secara default, kernel bergerak di atas gambar dengan menggeser satu posisi pada satu waktu secara horizontal atau vertikal. Dimulai pada posisi (0,0) bayangan dan jika langkahnya 1x1 maka akan bergerak 1 ke dua arah, horizontal atau vertikal. Langkah didefinisikan sebagai [1,2,2,1], itu berarti, setiap elemen dalam larik didefinisikan sebagai (batch, bergeser ke arah vertikal, bergeser ke arah horizontal, saluran). Berikut adalah contoh *strides*.



Gambar 2.9 Strides 2x2 digunakan untuk kernel dalam proses konvolusi pada ukuran gambar 6x6.

Pada konvolusi di atas, *strides* yang digunakan adalah 2x2, pada gambar 13b, *kernel* digerakkan oleh 2 posisi seperti yang ditunjukkan dalam warna merah pada arah *horizontal* dan pada gambar 2.9, kernel digerakkan oleh 2 posisi pada arah vertikal.

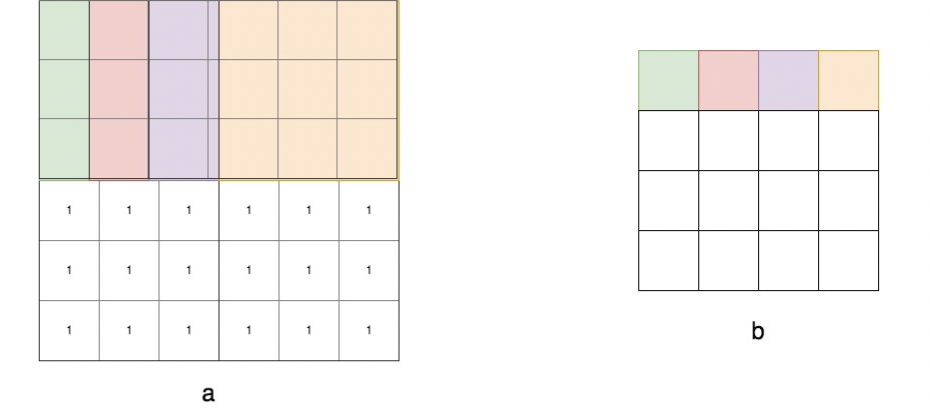
## 2.6.9 *Padding*

*Padding* pada gambar masukan adalah 3D atau 2D dengan angka nol, sehingga lapisan konvolusi tidak mengubah dimensi spasial gambar masukan. Dengan bantalan nol sementara konvolusi mengontrol ukuran spasial gambar keluaran dari lapisan konvolusi.

Padding dapat dihitung sebagai P = (F-1) / 2.

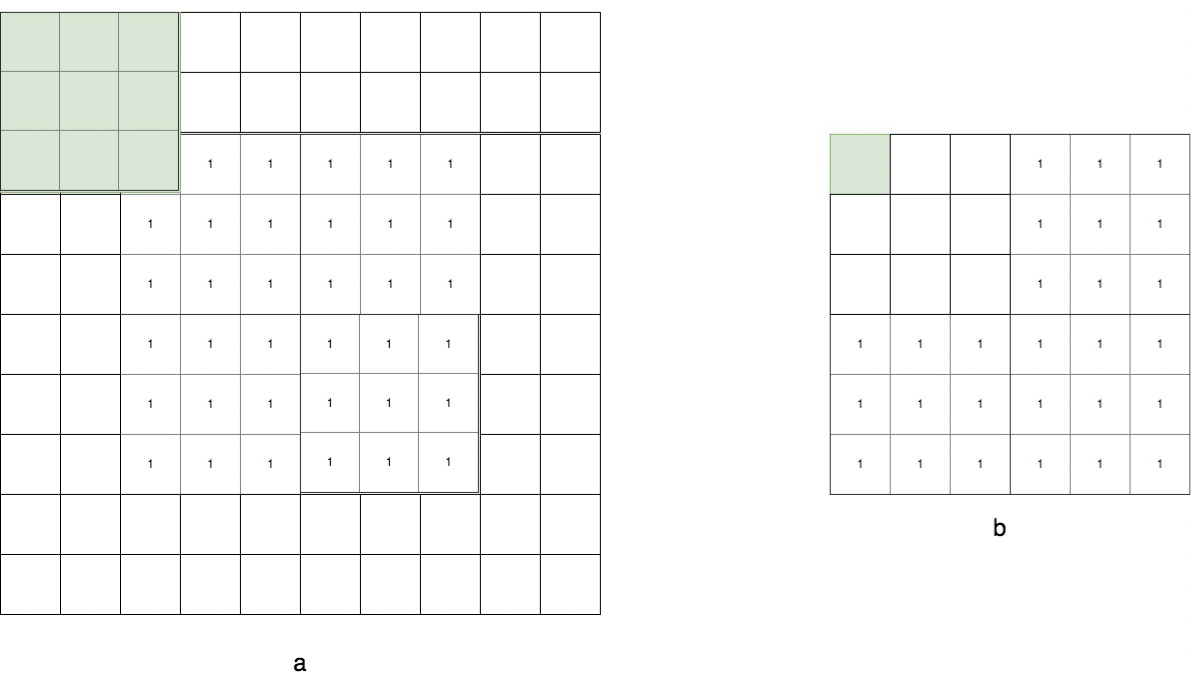
F: Ukuran filter

Berikut adalah contoh konvolusi tanpa padding atau padding = 0



Gambar 2.10 Konvolusi Zero Padding pada citra 6x6 yang mereduksi dimensi spasial.

Pada Gambar 2.10, konvolusi terjadi tanpa padding yang berakibat pada reduksi dimensi spasial. Karena itu, sebagian besar proses konvolusi kehilangan informasi dalam gambar. Jadi padding ditambahkan dalam proses konvolusi untuk menghindari masalah ini.



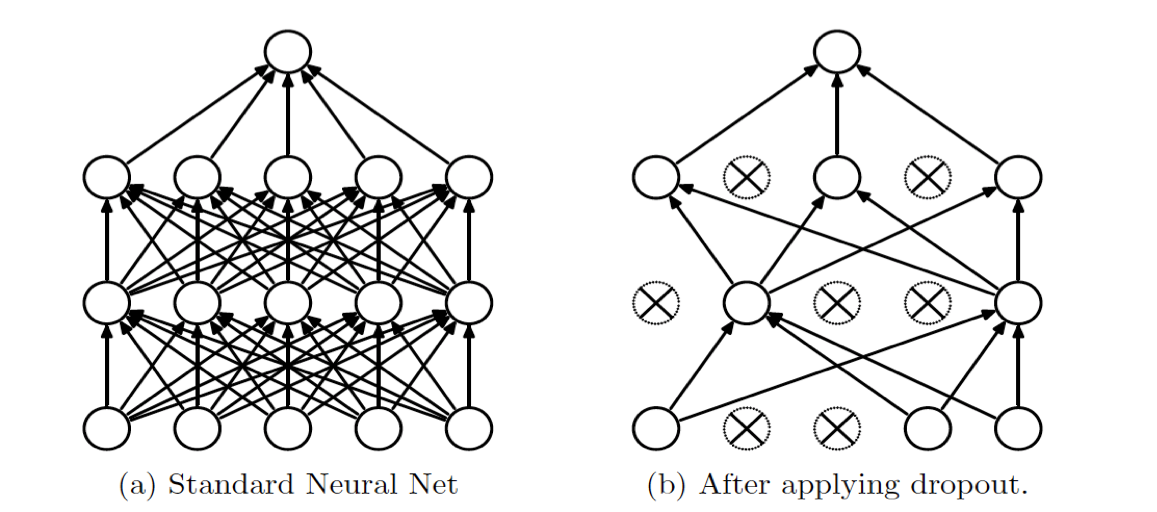
Gambar 2.11 Padding 2 piksel sebelum konvolusi pada gambar 6x6 untuk menjaga dimensi spasial.

Pada gambar 2.11, sebelum proses konvolusi, padding ditambahkan di sekitar citra ukuran 2, yang membantu menjaga dimensi spasial citra setelah konvolusi.

## 2.6.10 *Dropout*

Jaringan *deep learning* dengan banyak node dan *hyper-parameter* membuat model ini sangat disarankan dalam algoritma pembelajaran mesin. Akan tetapi, *overfitting* merupakan masalah serius dalam jaringan semacam ini. Jaringan yang besar akan sangat lambat, sehingga sulit untuk menangani *overfitting* dengan menggabungkan prediksi banyak jaringan saraf yang besar yang berbeda pada waktu pengujian. *Dropout* adalah teknik untuk mengatasi masalah ini. Untuk mengurangi *overfitting*, dalam laporan ini akan menerapkan *dropout* sebelum layer *readout*.

“*Dropout* adalah cara mencegah *overfitting* serta mengoptimalkan pembelajaran mesin” (Abhirawa, Jondri and Arifianto 2017, 4909). *Overfitting* merupakan sebuah indikasi setiap citra atau input yang telah diproses oleh pembelajaran telah mencapai titik optimal, tapi terjadi ketidak sesuaian saat proses pemrediksian. Lapisan *Dropout* sementara menghapus beberapa neuron di lapisan jaringan yang tersembunyi atau hanya terlihat dalam jaringan.

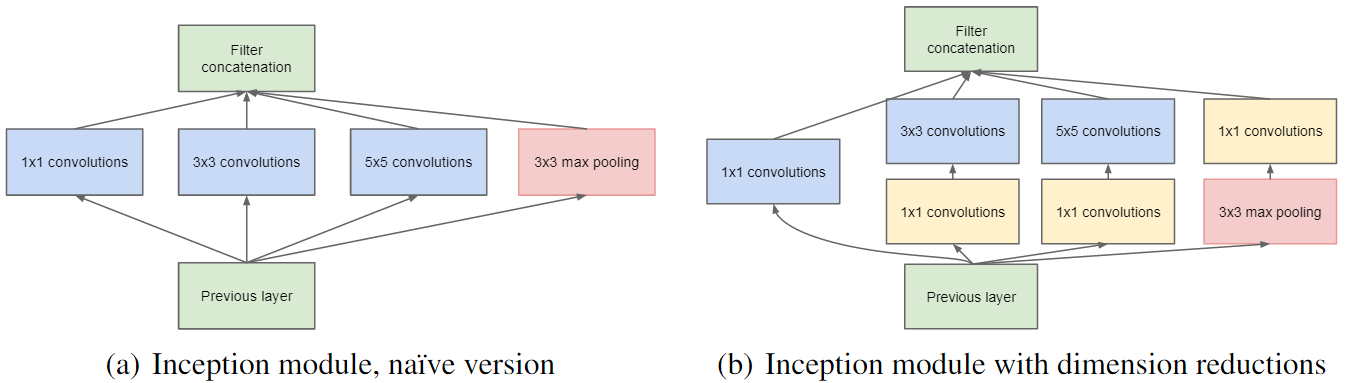


Gambar 2.12 Diagram Blok untuk operasi dropout.

## 2.6.11 *Inception Layer*

Arsitektur ini dikembangkan untuk mempercepat komputasi dengan fitur reduksi dimensi pada perancangan jaringan syaraf tiruan. Model *Inception* ini merupakan terobosan yang terjadi pada makalah *ImageNet*. Model ini memiliki beberapa konvolusi dengan beberapa filter dan lapisan penggabungan secara paralel di dalam lapisan yang sama yang disebut lapisan awal. Dalam arsitektur yang ditunjukkan pada gambar 13-a adalah model awal dasar dengan lapisan paralel, menggunakan konvolusi dengan filter 1x1 serta filter 3x3 dan 5x5 dan lapisan penggabungan maksimal. Gambar 13-b menunjukkan pengurangan dimensi dalam peta fitur dan menjelaskan penggunaan filter konvolusi 1x1 dapat membantu dalam pengurangan dimensi (karena jumlah saluran berkurang).

Tujuan di balik jenis arsitektur ini adalah untuk memungkinkan jaringan saraf dalam mempelajari lebih dalam dan menemukan bobot terbaik saat melatih model. *Model* ini secara otomatis memilih fitur yang berguna dan menggabungkan semuanya, sebelum meneruskannya ke lapisan berikutnya. Selain itu, hal ini dimaksudkan untuk mengurangi jumlah dimensi peta fitur, sehingga jumlah unit dan lapisan dapat ditingkatkan pada tahap selanjutnya.



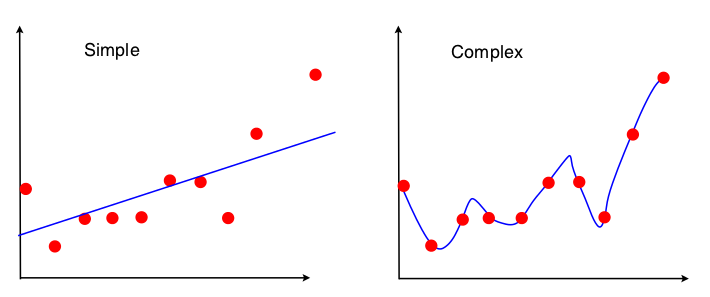
Gambar 2.13 Diagram blok reduksi dimensi naif dan permulaan.

## 2.6.12 Normalisasi

Umumnya, normalisasi data membantu mempercepat proses pelatihan. Jika kita tidak menormalkan data masukan, rentang distribusi data nilai fitur kemungkinan akan berbeda untuk setiap masukan. Normalisasi penting dalam hal klasifikasi untuk menghindari *overfitting* data. Kami menormalkan data dari nilai -127 hingga 127-piksel agar seragam di semua gambar skala abu-abu sebelum memulai proses pelatihan untuk kedalaman 1. Untuk pelatihan gambar kedalaman 3 (RGB), proses normalisasi melalui proses pengurangan nilai rata-rata piksel dari gambar dan membagi deviasi standar dari rata-rata wajah untuk membawa distribusi normal ke piksel gambar.

## 2.6.13 Regularisasi

Tujuan dari pembelajaran mendalam atau algoritma pembelajaran mesin adalah untuk mengabaikan *noise*. Dalam data yang *noise*, algoritma yang dibangun akan menjadi *overfitting* saat menyesuaikan fitur atau pola pada *noise* tersebut. Karena *noise* bersifat stokastik dan tidak menggeneralisasi data yang tidak terlihat menyebabkan kesalahan prediksi saat training. Hal ini menyebabkan tingginya akurasi prediksi pada data training, akan tetapi akurasinya sangat rendah saat memprediksi data yang tidak ada dalam training seperti *testing* dan data *validation.*



Gambar 2.14 Grafik kesesuaian fungsi regularisi.

Pada gambar di atas, pertama, garis lurus mencoba untuk memasukkan sebagian besar titik yang memungkinkan, namun grafik kedua menunjukkan semua titik sesuai dengan polinomial derajat K. Di sisi kanan, ini dicocokkan dengan noise, dengan cara menarik garis derajat polynomial yang lebih tinggi sehingga mengurangi kesalahan secara signifikan dibandingkan dengan yang lebih sederhana di sisi kiri. Metode ini disebut dengan regularisasi. Untuk menghindari *overfitting* dan jangkauan bobot maksimal, diperlukan melakukan regularisasi untuk menjaga bobot tetap di bawah batas ambang.

## 2.6.14 Optimasi

Dalam optimasi *deep learning* terdapat banyak konteks. Terkadang pengoptimalan analitik memang membantu dalam penulisan bukti ilmiah dari hasil pembelajaran mesin. Masalah pengoptimalan yang paling banyak dalam *deep learning* adalah pelatihan jaringan saraf tiruan. Pengoptimalan adalah proses menemukan parameter pengoptimalan terbaik yang dapat dikurangi secara signifikan oleh *cost function* pada jaringan saraf tiruan. Ada banyak pengoptimal yang tersedia seperti *Adam Optimizer* atau *Gradient Descent* untuk mengoptimalkan fungsi kerugian, ketika ada fitur bagus yang ditemukan dan mencapai titik optimal dan minimum globalnya. Beberapa pengoptimalan dijelaskan sebagai berikut.

### 2.6.14.1 Pengoptimalan Adam (*Adam Optimizer*)

Pengoptimal Adam menggunakan algoritma Kingma dan Ba's Adam untuk mengontrol kecepatan pembelajaran mesin. Adam memiliki beberapa keunggulan dibandingkan *Gradient Descent Optimizer* yang sederhana. Terutama saat menggunakan parameter *mean moving* (momentum).

Hal ini memungkinkan pengoptimal untuk menggunakan ukuran langkah efektif yang lebih besar, dan algoritma akan menyatu dengan ukuran langkah ini tanpa penyempurnaan. hal ini membutuhkan lebih banyak komputasi yang harus dilakukan untuk setiap parameter di setiap langkah pelatihan (untuk mempertahankan rata-rata bergerak dan varians serta menghitung gradien berskala), dan lebih banyak status yang dipertahankan untuk setiap parameter.

### 2.6.14.2 Pengoptimal Penurunan Gradien

Pengoptimal Penurunan Gradien (*Gradient Descent*) sederhana dapat dengan mudah diimplementasikan, akan tetapi membutuhkan lebih banyak penyesuaian *hyperparameter* sebelum dapat menyesuaikan dengan cepat. Selain itu beberapa fungsi berikut dapat digunakan untuk mengoptimalkan fungsi biaya (*cost function*).

1. Pengoptimalan *Adadelta*
2. Pengoptimalan DAO *Adagrad*
3. Pengoptimalan *Adagrad*
4. Pengoptimalan *Penurunan Gradien*
5. Pengoptimalan *Adam*

## 2.6.15 Fungsi *Loss*

Fungsi kerugian (*loss function*) merupakan bagian penting dalam jaringan *deep learning*, terkadang fungsi kerugian memperhatikan kesalahan dalam klasifikasi tetapi tidak dioptimalkan, yang digunakan untuk mengukur ketidak konsistenan antara nilai prediksi dan label aktual. Ini adalah nilai non-negatif yang keluar dari lapisan klasifikasi akhir, di mana kekuatan dan akurasi *model deep learnin*g meningkat seiring dengan penurunan nilai fungsi kerugian. Fungsi kerugian merupakan fungsi risiko empiris yang terdiri dari istilah risiko empiris dan istilah regularisasi.

1. *Mean Square Error*
2. *Mean Square Logaritmic Error*
3. L2
4. *Mean Absolute Error*
5. *Cross Entropy error*

## 2.6.16 Penghentian Awal (*Early Stopping*)

Penghentian awal adalah bentuk regularisasi yang digunakan untuk menghindari *overfitting* saat melatih model secara iteratif seperti penurunan gradien. Penghentian awal dihitung untuk membuka set validasi, dan kerugian yang ditimbulkan oleh jaringan pada setiap iterasi. Saat jaringan neural mengamati kerugian meningkat dari batas tertentu di terendah dan mencoba menghentikan pelatihan model dan menggunakan model kerugian terendah untuk pengujian. Metode ini juga menyediakan cara untuk menghentikan model saat mulai *overfitting*.

## 2.6.17 *One-hot Encoding*

Vektor dengan *one-hot encoding* merupakan proses dimana variabel yang telah dikategorikan dikonversikan ke dalam bentuk yang disediakan algoritma *machine learing*. Sebagai contoh apabila memiliki 2000 kelas untuk dilatih dan di lapisan terakhir, vektor yang berisi 2000 bilangan bulat dan 1999 adalah nol dan karena jaringan dilatih dengan 2000 kelas maka setiap kelas akan memiliki vektor 2000-dimensi untuk setiap node keluaran. Hal ini memiliki proses komputasi yang sangat tinggi jika dataset yang dimiliki sangat besar.

## 2.6.18 *Embedding*

Dari pada menggunakan satu h*ot encoding*, dapat menentukan ukuran matriks *embedding* untuk mempertahankan berapa banyak nilai *embedding* yang kita perlukan untuk mengklasifikasikan satu label kelas untuk tujuan klasifikasi. Nilai penyematan adalah nilai pilihan terbaik untuk setiap label kelas dan, untuk masukan yang tidak diketahui di masa mendatang, gambar dapat diklasifikasikan dengan mengukur jarak dari setiap kelas nilai penyematan ke nilai *embedding* gambar baru.

*Embedding* umumnya merupakan representasi vektor dari sebuah gambar dalam ruang dimensi fitur tertentu. Jika kita mendefinisikan vektor *embedding* dari ruang 128-dimensi. Kemudian setelah beberapa konvolusi dan kumpulan maksimul dan ketika datang ke lapisan *fully connected layer* akan meratakan (*flattening*) matriks gambar ke vektor fitur dan mengubahnya menjadi 128 ukuran dan ini dapat digunakan untuk mewakili gambar masukan dalam ruang 128-dimensi.

## 2.6.19 Pemrosesan *Batch*

Pemrosesan *batch* adalah teknik pemrosesan beberapa secara bersamaan dalam pelatihan. Sebagian besar CNN dan arsitektur jaringan *deep learning* lainnya mendukung pemrosesan batch melalui *Tensorflow*. Pemrosesan batch membutuhkan daya komputasi untuk memproses dan melakukan komputasi matematis sekaligus pada matriks berukuran besar. Misalnya, jika memiliki gambar 120x120 dan telah memilih ukuran tumpukan sebagai 100 maka *Deep Learning Networks* akan memproses input sebagai 100x120x120 secara bersamaan. Untuk itu banyak memori dan unit GPU yang harus ditempatkan untuk menangani sejumlah operasi matematika. Ini sangat efisien dalam menghitung dan memperbarui vektor bobot. Dalam propagasi mundur, bobot yang terkait dengan setiap *node* diperbarui satu kali dalam satu batch dari pada memperbarui untuk setiap gambar saat kami memproses dalam algoritma penurunan gradien stokastik.

## 2.6.20 *Region-Based Convolutional Neural Network*

Algoritma ini merupakan metode untuk mendeteksi objek dalam lingkup visi komputer berdasarkan jaringan konvolusi. R-CNN sendiri pada awalnya dikembangkan sebagai cara atau metode untuk mengenali citra dengan menggabungkan algoritma *Convolutional Neural Network* dengan *Region Proposed Network*. Metode R-CNN dikembangkan dan ditingkatkan kinerjanya baik dalam kecepatan dan ketepatan Ketika proses pengidentifikasian objek. Namun, R-CNN dan beberapa model turunanannya seperti *Fast* R-CNN memiliki kelemahan, seperti *bottleneck* pada salah satu model. Dikatakan tidak sebanding dengan kecepatan komputasi yang dimiliki Jaringan konvolusi. Jaringan konvolusi lebih cepat telah dapat mengoptimalkan penggunaan fungsi konvolusi untuk mempercepat proses RPN dan mengurangi kemacetan (*bottleneck*). “Metodenya dengan menggabungkan pendeteksian objek berbasis deep learning agar berjalan di *frame rate* yang mendekati real time. Lalu *learned* RPN juga meningkatkan kualitas region proposal serta keakuratan pendeteksian objek secara keseluruhan” (Ren, et al. 2015, 12).

## 2.6.21 Model Jaringan CNN

Sebuah jaringan CNN terdiri dari lapisan yang lebih dari satu. Setiap lapisan bertanggung jawab untuk mengambil input dan memprosesnya menggunakan fungsi spesifik yang didefinisikan di lapisan itu untuk menjadi keluaran. *Stack* lapisan ini dinamai sebagai CNN *models*.

Beberapa publikasi mengenai jaringan CNN telah banyak yang memasuki popularitas penggunanya. Yang paling umum digunakan biasanya *FastNet*, *FasterNEt*, dan *ResNet*. Biasanya, sebuah model diuji kemampuannya dalam proses pendeteksian objek.

## 2.7 *Face Recognition*

Pengenalan wajah adalah tugas rutin dan mudah yang dilakukan orang pada keseharian mereka. Riset dalam pengidentifikasian muka telah popular dengan signifikan berdasarkan kehadiran komputer berperforma tinggi serta sistem tertanam, mendorong minat yang luar biasa dalam pemrosesan gambar serta film digital. “…motivasi riset dari pengidentifikasian wajah adalah pemrosesan citra gambar maupun *video* secara otomatis termasuk diantaranya interaksi manusia dan komputer, autentikasi citra bio serta pemantauan.” (Li and Jain 2005, 1).

Pengenalan muka merupakan visi komputer yang mana komputer memperoleh identifikasi dan informasi pribadi dari gambar wajah dengan menganalisis gambar wajah dalam gambar dan membandingkannya dengan data gambar wajah yang sebelumnya disimpan dalam database. Deteksi wajah biasanya dilakukan dari depan, dengan pencahayaan seragam di seluruh wajah. Namun, ada juga masalah seperti posisi wajah, ukuran wajah serta jarak wajah, orientasi gambar, usia objek, dan ekspresi wajah. “Secara umum, sistem pengenalan wajah meliputi empat hal: pengenalan, pendaftaran, pengambilan ciri objek, serta pengidentifikasian. Proses penormalan (serta perataan wajah) merupakan tahapan sebelum proses deteksi wajah (ekstraksi dan pencocokan fitur wajah) dilakukan.” (Li and Jain 2005, 2).

Pendeteksian wajah merupakan tahap pertama menuju pengenalan wajah atau *facial recognition*. “Detektor wajah idealnya harus dapat mengenali dan menunjukkan posisi semua lokasi serta area wajah dalam gambar tanpa memandang skala, ekspresi, sudut pandang serta usia.” (Li dan Jain 2005, 13).

*Face recognition* memberikan batasan posisi dan skala untuk setiap data citra wajah yang terdeteksi, sehingga proses *alignment* atau penyelerasan ini bertujuan untuk meningkatkan akurasi lokasi citra wajah dan normalisasi. Setelah dilakukan normalisasi pada wajah, selanjutnya dilakukan ekstraksi fitur untuk mendapatkan data yang valid agar membantu memisahkan citra data wajah dari orang yang bersangkutan. Cukup stabil untuk berbagai data geometrik dan fotometrik dan berbeda satu sama lain. Metode identifikasi wajah diterapkan dengan mencocokkan ciri-ciri dari objek yang diambil melalui informasi wajah *input* terhadap informasi wajah pada data pelatihan dan data eksperimen yang disimpan sebagai basis data wajah.

## 2.8 Kerangka Kerja *Deep Learning*

Kerumitan dalam penulisan kode sumber pada program *Graphic Processing Unit* (GPU) dengan performa lebih optimal membuat peneliti memerlukan untuk memberikan definisi struktur alur kerja atau *framework* guna mencegah repetisi pada saat pembuatan kode sumber pembelajaran mesin baru. Untuk menghindari hal tersebut, diperlukan pembuatan pustaka piranti lunak yang bisa memproses membutuhkan kinerja tinggi, di antaranya: *convolutional processing operations* serta *matrices pow*. Keduanya dapat dijabarkan pada suatu kerangka kerja untuk menjalankan proses menggunakan pustaka yang disebutkan. (Goodfellow, Bengio and Courville 2016, 446). Metode ini memberikan kemudahan serta mendukung berbagai macam piranti keras. Misalnya, program yang dibuat dengan kerangka kerja Theano dapat berjalan di Dapur Proses Pusat (CPU) serta Dapur Proses Grafis (GPU) tanpa melakukan perubahan pada kode sumber.

Pada dilaksanakan penelitian, sudah terdapat bermacam framework pembelajaran mendalam yang menawarkan kemampuan serupa, antara lain *Theano, Keras, TensorFlow,* dan *PyTorch.* Semua kerangka atau kerangka kerja yang disebutkan memiliki tujuan umum untuk memproses sejumlah besar data intensif komputasi dan melibatkan sejumlah besar proses berulang. Setiap kerangka pembelajaran mendalam terus berkembang dengan mengoptimalkan berbagai aspek seperti algoritma yang bermacam-macam, dukungan untuk arsitektur pembelajaran mendalam yang berbeda, penulisan dan pemodelan kode sumber yang lebih mudah, dan kecepatan komputasi yang lebih cepat.

## 2.8.1 Tensorflow

Tensorflow adalah kerangka kerja pembelajaran mesin yang dibuat secara sumber terbuka yang dilatar belakangi oleh *Alphabet Inc*. “Tensorflow dapat memberikan pengalaman yang memudahkan bagi pengembang untuk melakukan eksperimen dengan algoritma optimasi dan pelatihan baru” (Abadi 2016, 1). Pada pengaplikasiannya, Tensorflow telah memberikan dukungan terhadap berbagai macam jenis algoritma pembelajaran mesin, termasuk di antaranya proses pelatihan dan inferensi untuk berbagai jenis model *Deep Neural Network*.



Gambar 2.15 Logo Tensorflow.

Berikut beberapa fitur utama yang dimiliki Tensorflow.

* + - 1. Mendefiniskan, mengoptimalkan dan mengkalkulasi ekspresi matemastis yang melibatkan array multi dimensi (tensors) secara efisien.
      2. Mendukung pemrograman Algoritma Pembelajaran mesin berupa *Deep Learning*.
      3. Transparansi penggunaan sumber daya kartu grafis serta cukup baik dalam optimisasi penyimpanan sementara, sehingga Tensorflow dapat mengidentifikasi pada komputasi manakah yang memerlukan ditangani oleh kartu grafis.
      4. Keterluasan dan reliabilitas pada pemrosesan data dan komputasi yang sangat tinggi.

## 2.8.2 Keras

Keras dapat dijabarkan sebagai pustaka yang dapat berjalan di atas kerangka kerja pembelajaran mesin seperti Tensorflow. “Semenjak dirilisnya Tensorflow versi 2, Keras telah diadopsi sebagai standar high level API, lebih besar dan membuat pemrograman menjadi lebih intuitif” (Guilli, Kapoor and Pal 2019, 3). Framework ini sendiri ditulis dengan menggunakan Bahasa Pemrograman Python. Keras menyediakan antar muka level tinggi untuk memudahkan pengguna mengakses library yang disediakan *framework machine learning*. Pustaka ini bisa dijalankan baik menggunakan CPU serta Kartu Grafis. Selain itu, pustaka ini juga mendukung permodelan lain yang berhubungan dengan Jaringan Saraf Tiruan seperti *convolution, pooling, recurrent, embedding*, dan lain-lain.



Gambar 2.16 Logo Keras.

## 2.9 Python

Python merupakan bahasa pemrograman yang dikembangkan Guido Van Rossum. Dia adalah seorang akademisi berdarah Belanda. Awalnya, pembuatan bahasa ini melibatkan pembuatan kode sumber untuk *high-level programming language* pada *operating system* yang berjalan secara distribusi. Bahasa pemrograman ini dilibatkan oleh banyak pengolah piranti lunak serta diadaptasi oleh banyak instansi guna menciptakan piranti lunak komersil. Namun, dengan bahasa pemrograman Python, terdapat hal yang bisa ditemukan. Beberapa paket paling populer di Python adalah:

1. *Flask*, merupakan kerangka kerja untuk membuat aplikasi berbasis web.

2. *Scikit-Learn*, piranti lunak digunakan menciptakan kecerdasan buatan serta system pembelajaran mesin.

3. *Zero Messages Queue* (ZMQ), pustaka untuk membuat aplikasi antrian data, serta *task* yang tidak berurutan (*asynchronous*).

4. OpenCV Python adalah Pustaka yang digunakan untuk membuat system berbasis penginderaan komputer.

5. TensorFlow adalah Pustaka yang digunakan untuk mengembangkan system dengan algoritma pembelajaran mendalam, dan lain-lain.

Python juga memiliki sebuah manajer paket yang terkenal dikalangan pengembang yang disebut Pip. Dengan menerapkan pip sebagai manajer pustaka, pengguna memiliki keleluasaan untuk memasang maupun menghapus pustaka Python yang akan / atau tidak dipakai kembali dengan lebih mudah.

## 2.10 MySQL

Basis data adalah kumpulan data yang disusun untuk menciptakan informasi yang berguna. Basis data terdiri atas sekumpulan *row* dengan tipe data serupa. Umumnya, pengembang piranti lunak menggunakan RDBMS dengan Bahasa SQL. MySQL merupakan piranti lunak sistem manajemen basis data relasional.

*Structured Query Language* atau biasa disingkat SQL, merupakan kode perintah tertentu ditujukan sebagai cara mengakses dan mengatur operasi tambah, hapus, ubah dan baca sebuah basis data. Oleh karena itu, MySQL dan SQL bukanlah hal yang sama. Jadi MySQL adalah perangkat lunak, sedangkan SQL adalah bahasa perintah yang digunakan di dalamnya. Saat membandingkan MySQL dengan sistem basis data lain, pengguna harus mempertimbangkan apa yang paling penting saat membutuhkannya. Pertanyaan yang biasa muncul adalah mengenai penampilan, dukungan, fungsionalitas SQL, persyaratan keamanan kontrak maupun pembayaran. Dengan pertimbangan ini, MySQL mempunyai beberapa fitur seperti:

1. Karena kecepatannya, banyak ahli yang berpendapat bahwa MySQL adalah server basis data yang paling cepat.

2. MySQL adalah database berkinerja tinggi namun sederhana yang mudah diatur.

3. MySQL saat ini gratis untuk tujuan non komersial.

4. MySQL memahami SQL (Structured Query Language), sistem basisdata pilihan.

5. Beberapa konektor bisa terhubung secara bersamaan. Beberapa *database* dapat digunakan secara bersamaan.

6. Anda dapat mengakses database MySQL Anda dari mana saja melalui Internet.

7. Basi Data ini dapat dijalankan di *multi-platform*.

8 MySQL sudah tersedia serta memiliki kode sumber yang dapat didistribusikan untuk pengembangan lebih lanjut.

9. Dapat terhubung ke berbagai macam Bahasa Pemrograman termasuk Python.

MySQL juga memiliki kekurangan sebagai berikut:

1. MySQL tidak mendukung koneksi ke beberapa bahasa pemrograman yang berbasis visual. Sambungan ini berarti bahwa bidang yang dimuat harus sesuai dari program visual.

2. Ukuran row yang diolah relatif kecil.

3. Sistem data terdistribusi belum dapat didukung.

## 2.11 Flask

Flask adalah web microframework Python, pustaka Bahasa Pemrograman Python dari pihak ketiga yang Digunakan untuk Mengembangkan Layanan Web. “Flask adalah framework yang sederhana, tetapi sangat mudah untuk dikembangkan” (Relan 2019, 1). *Flask* memberikan kemudahan pada pengembang perangkat lunak untuk membuat web terstruktur dan mengelola perilaku web dengan lebih mudah. Flask termasuk dalam tipe micro-framework karena tidak memerlukan penggunaan tools atau library tertentu.

Fitur dan komponen yang paling umum seperti validasi formulir, *database*, dll. tidak diinstal di Flask secara *default*. Flask menyebut dirinya sebagai kerangka mikro, tetapi tidak kekurangan fitur. *Micro-framework* di sini berarti Flask bertujuan untuk menjaga inti dari aplikasi sesederhana mungkin sambil melakukan penambahan dengan mudah. Seperti yang terlihat, fleksibilitas dan skalabilitas Flask cukup tinggi dibandingkan dengan kerangka kerja lain.

*Flask* memiliki dua komponen utama yang bernama *Werkzeug* dan *Jinja*. “*Werkzeug* memberikan fitur seperti *routing, debugging*, serta *Web Server Gateway Interface* (WSGI), Lalu *Flask* menggunakan *Jinja2* sebagai *template engine*” (Relan 2019, 1). Komunitas pengembang *Flask* sendiri relatif besar dengan ribuan extension yang bersifat *opensource*. Hingga penelitian ini ditulis, *Flask* telah merilis versi 1.1.2.

## 2.12 *Representational State Transfer Application Programming Interface* (RESTful-API)

“*RESTful-API* merupakan gaya arsitektur perangkat lunak untuk layanan berbasis web yang menyediakan standar komunikasi diantara sistem yang berbeda-beda” (Relan 2019, 4). Namun, ada juga RESTful API, protokol standar yang didukung untuk melakukan REST yang disertai fiturAmbil*,* Kirim*,* Ubah*,* Hapus*,* danUbah-sebagian. Jadi RESTful dipastikan REST, akantetapi REST tidak merepresentasikan RESTful.

## 2.12.1 *Representational State Transfer* (REST)

REST adalah bagaimana sebuah aplikasi berbasis web membuat standar untuk saling berkomunikasi. Biasanya, protocol yang digunakan adalah *Hypertext Transfer Protocol* (HTTP). Metode ini dikembangkan pada tahun 2000 oleh Roy Fielding. Server REST secara arsitektur akan memberikan data (atau sumber daya) lalu klien REST akan mengkonsumsi sumber daya ini serta menyajikannya agar digunakan lebih lanjut. Seluruh sumber daya dikenali oleh URI ​​atau domain utama. Sumber daya yang ditampilkan dalam format teks.

Bentuk umum jenis data yang dipakai adalah JSON dan XML. Berikut kata kunci yang sering diimplementasikan pada REST.

a. *Dapatkan*, Membaca atau mengambil sumberdaya suatu informasi.

b. *Ambil* Diimplementasikan sebagai cara memperbarui data.

c. *Hapus* Diimplementasikan sebagai cara menghapus sumber daya.

d. *Kirim* sebagai cara memasukkan data ke dalam badan *request* untuk membuat data baru ketika permintaan dibuat.

e. *Opsional* digunakan untuk memperoleh operasi yang didukung oleh sumber daya contoh.

## 2.12.2 *Application Programming Interface* (API)

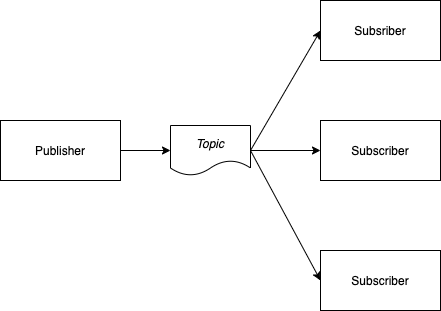
API merupakan bentuk penyingkatan dari *Application Programming Interface*. Kata kunci ini merupakan sebuah tautan yang memberikan kemudahan bagi sistem agar dapat saling berkomunikasi dengan sistem lain dengan format informasi berbeda.

## 2.13 *Message Queue*

*Message Q*ueue atau juga dikenal dengan sebutan *Message Broker* adalah sebuah *broker* yang menyediakan komunikasi yang bersifat *asynchronous* antar sistem. Hal ini dimaksudkan agar sistem yang dibangun tidak selalu menunggu keseluruhan proses selesai terlebih dahulu (*asynchronous ta*sk), sehingga hasil proses yang diterima pengguna tidak berlangsung terlalu lama. Pesan (*topic*) yang diproses oleh *Message Queue* berbentuk *Queue* atau antrian, yang berarti pesan yang pertama kali diterima oleh *Message Queue* akan diproses terlebih dahulu secara berurutan atau biasa dikenal dengan sebutan mekanisme *First In First Out (FIFO)*. Untuk pesan yang dikirim oleh *Message Queue* sendiri berbentuk *key-value pair.* Artinya adalah terdapat dua bagian yakni *key* sebagai kata kunci dari pesan yang bersifat unik atau tidak akan sama dengan pesan yang lain. Sedangkan value adalah isi pesan yang dikirim. Standar layer aplikasi pada *protokol* MQ disebut *Advanced Message Queuing Protocol* (AMQP). Me*ssage Queue* memiliki dua mekanisme dalam memroses pesan.

## 2.13.1 Mekanisme *Pub/Sub*

Dalam mekanisme *Pub/Sub*, terdapat dua bagian sistem yakni *Publisher* dan *Subscriber*. *Publisher* bertindak sebagai pengirim pesan, sedangkan *Subscriber* bertindak sebagai penerima pesan. Pada umumnya setiap *Publisher* memiliki banyak *Subscriber*. Hal ini bertujuan agar setiap pesan yang dikirim tidak hanya dikonsumsi oleh satu penerima saja, melainkan banyak sistem yang memiliki tujuannya masing-masing.



Gambar 2.17 Mekanisme Pub/Sub.

Contoh kasus yang dapat diselesaikan menggunakan mekanisme ini adalah sebuah sistem notifikasi pendaftaran. Apabila pengguna telah selesai melakukan pendaftaran, maka akan muncul notifikasi bahwa pendaftaran telah berhasil dilakukan di beberapa *platform* secara *asynchronous* seperti email, tampilan di *web*, dan SMS. Hal ini dapat dilakukan karena sistem utama yang menangani pendaftaran bertindak sebagai *Publisher* pesan yang berisi pendaftaran telah berhasil dilakukan. Sedangkan SMS, *Web* dan *Email* bertindak sebagai penerima pesan atau *Subscriber* yang memroses pesan tersebut.

## 2.13.2 Mekanisme *Req/Rep*

Mekanisme *Req/Rep* memiliki bentuk yang lebih sederhana yakni setiap satu pengirim memiliki satu penerima. Pengirim disebut *Requester* sedangkan penerima disebut *Repplier*. Mekanisme ini lebih dikhususkan untuk sistem yang membutuhkan *reliability* yang tinggi untuk pengguna. Konsep yang ditawarkan adalah mirip seperti pada HTTP *Request-Response* biasa akan tetapi dalam bentuk socket sehingga komunikasi yang terjadi bisa mendekati waktu sekarang.

## 2.13.3 *ImageZMQ*

“*ImageZMQ* adalah sebuah pustaka yang secara khusus menangani gambar dari pustaka OpenCV dari satu komputer ke komputer lain menggunakan *Message Queue* bernama PyZMQ” (Bass 2020). Penggunaan ImageZMQ mempermudah pengembang untuk membuat sistem yang membutuhkan mengirim gambar secara stream dan dalam waktu nyaris sekarang. *ImageZMQ* mendukung kedua mekanisme pengiriman pesan yang dimiliki *Message Queue* yakni mekanisme *Pub/Sub* dan *Req/Rep.* Selain yang disebutkan, kelebihan yang ditawarkan dibanding dengan pustaka *Message Queue* yang lain adalah sebagai berikut.

1. *ImageZMQ* menggunakan protokol *peer to peer* yang berarti tidak membutuhkan *server* secara khusus untuk dijadikan tempat komunikasi antar sistem.
2. *ImageZMQ* memiliki *throughput* yang besar diantara pengirim dan penerima pesan.

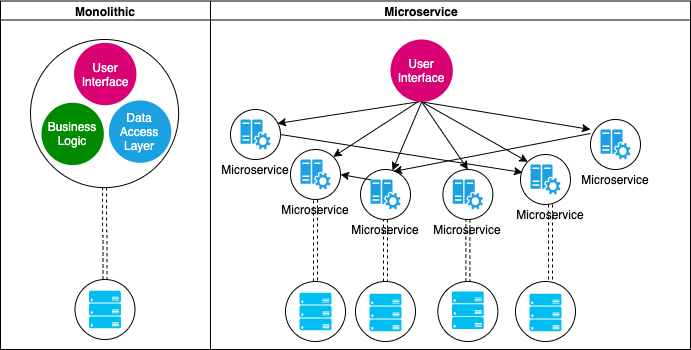
## 2.14 Arsitektur *Microservice*

Dalam membangun arsitektur perangkat lunak, diperlukan analisis di mana pengembang menentukan solusi yang optimal untuk membangun struktur sistem yang dapat memenuhi kebutuhan. Baik kebutuhan secara performa, keamanan maupun pengembangan keberlanjutan dari sistem itu sendiri. Dari segi prinsip dan paradigma, terdapat dua macam arsitektur dari sebuah sistem. Yakni arsitektur *Monolithic* dan *Microservice*.

*Monolithic* merupakan sebuah sistem yang dibagun di mana keseluruhan proses dan elemen sistem berada pada satu *server*. Hal ini meminimalisasi kemungkinan terjadinya masalah komunikasi antar modul dari sistem yang dibagun. Akan tetapi apabila modul yang dibagun sangat besar dengan kompleksitas yang tinggi atau membutuhkan komputasi yang tinggi, maka performa dari aplikasi yang dibagun akan menurun drastis dikarenakan adanya batasan kemampuan komputasi dari *server* yang digunakan. Selain itu *monolithic* kurang mampu menangani pekerjaan yang bersifat paralel yang berarti bahwa keseluruhan proses dilakukan oleh satu *server* saja dalam bentuk antrian proses yang saling *interrupt*.

“*Microservice* adalah servis yang dapat diterbitkan secara independen yang dibuat berdasarkan domain bisnis. Mereka saling berkomunikasi satu sama lain melalui jaringan (Newman 2020, 1).” Dari pengertian tersebut dapat disimpulkan *microservice* harus memiliki kelebihan-kelebihan sebagai berikut.

1. *Microservice* dibagun dengan membagi sistem menjadi modul kecil dan didistribusikan pada tiap-tiap *server* yang berarti bahwa penanganan masing-masing modul dilakukan oleh satu atau beberapa *server*.
2. Modul yang dibagun harus merepresentasikan sebuah domain bisnis yang akan dibuatkan solusinya.
3. Modul yang terbagi pada sistem terdistribusi harus dapat bekerja sendiri (independen) tanpa harus bergantung pada *service* atau modul yang lain.
4. Masing-masing *service* saling berkomunikasi menggunakan jaringan komputer.



Gambar 2.18 Contoh perbedaan arsitektur Monolithic dan Microservice.

Dari kemampuan fleksibilitas dan efisiensi yang dimiliki oleh arsitektur ini, sistem dapat berjalan dengan relatif lebih cepat sehingga tidak diperlukan super komputer layaknya arsitektur *monolithic* untuk menangani proses yang sedang dieksekusi.