BAB II
TINJAUAN PUSTAKA

2.1 Penelitian Sebelumnya

Beberapa penelitian terdahulu telah menggunakan *Transfer Learning* CNN untuk mengklasifikasi penyakit tanaman singkong. Sebuah penelitian berjudul "*A Mobile-Based Deep Learning Model for Cassava Disease Diagnosis*". Penelitian ini menggunakan *Single Shot Multibox MobileNet*. Model ini dapat melakukan pelokalan objek dan klasifikasi objek dalam satu pemrosesan, dan ini merupakan komponen kunci dalam memberikan pengenalan objek waktu nyata pada perangkat seluler. *Dataset* yang digunakan sebanyak 2,145 yang terdiri dari 7 kelas daun singkong. Penelitian ini menghasilkan keluaran berupa aplikasi mobile yang dapat mengklasifikasi penyakit daun singkong dengan akurasi mencapai 70,4% (Ramcharan A, dkk., 2019).

Selanjutnya penelitian dengan judul "*Deep learning for detection cassava leaf disease*". Penelitian ini menggunakan *pre-trained model MobileNetV2*. Model ini menggunakan fungsi aktivasi *interlayer*, yang memungkinkan lapisan nonlinier dari lapisan sebelumnya dilinierkan dan ditransfer sebagai input ke lapisan berikutnya. Model melanjutkan proses pelatihan hingga mencapai tingkat akurasi terbaik. Lalu *dataset* yang digunakan ada 5 jenis, yaitu *Cassava Bacterial Blight* dengan 466 gambar, *Cassava Brown Steak Disease* dengan 537 gambar, *Cassava Green Mite* dengan 450 gambar, *Cassava Mosaic Disease* dengan 548 gambar, dan *Healthy* dengan 316 gambar. Akurasi yang dihasilkan dari model CNN di penelitian ini adalah 65,6%. Lalu model tersebut *dideploy* pada *Python GUI*. (Ayu dkk., 2021).

Lalu selanjutnya penelitian dengan judul "Pengenalan Penyakit Pada Tanaman Pokok di Indonesia Dengan Metode *Convolutional Neural Network*". Pada penelitian ini juga terdapat tanaman singkong sebagai salah satu objek penelitiannya. Metode yang digunakan pada penelitian ini adalah CNN dengan arsitektur *Inception v3*. Model hasil pembelajaran dikonversi menjadi *TFLite,* sehingga dapat ditanamkan pada aplikasi berbasis *Android*. *Dataset* daun tanaman yang digunakan ada 4, yaitu 9148 gambar tanaman singkong, 7135 gambar tanaman

kentang, 9430 gambar tanaman singkong, dan 3355 gambar tanaman padi. Hasil program dari penelitian ini mengidentifikasikan penyakit pada tanaman singkong, kentang, singkong, dan padi. Hasil model klasifikasi pada lapangan menghasilkan rata-rata akurasi sebesar 65% (Angjaya dkk., 2021).

Penelitian dengan tujuan identifikasi penyakit tanaman singkong sudah beberapa kali dilakukan oleh para peneliti, namun perkembangan teknologi dari tahun ke tahun selalu menghadirkan metode dan algoritma terbaru dengan kemampuan yang lebih baik. Pendekatan *transfer learning* membuat pelatihan model menjadi lebih mudah dan cepat, bahkan mengurangi probabilitas *overfitting*. Namun dalam penerapannya ada banyak variasi modelnya, seperti beberapa penelitian terdahulu diatas yang memakai model *SSD MobileNet, MobileNetV2,* dan juga *Inception V3*.

Namun sayangnya hasil akurasi dari penelitian - penelitan menggunakan model tersebut masih berkisar antara 65% - 70%. Untuk itu penulis bermaksud untuk melakukan penelitian menggunakan model yang belum pernah digunakan pada klasifikasi penyakit singkong berdasarkan citra daun, yaitu *DenseNet121*. *DenseNet121* memodifikasi struktur jaringan CNN yang mana setiap layer akan memperoleh masukan tambahan dari lapisan sebelumnya atau dengan kata lain setiap lapisan baru akan menerima *feature map* dari lapisan-lapisan sebelumnya. Sehingga arsitektur ini mempunyai kemungkinan hasil akurasi lebih besar. Selain itu penulis juga akan men-*deploy* model ke aplikasi *android* agar lebih mudah digunakan di lapangan.

2.2 Penyakit Daun Singkong

Tanaman singkong atau ubi kayu, adalah pohon tahunan tropika dan subtropika dari turunan keluarga Euphorbiaceae. Singkong umumnya dijadikan olahan makanan karena nutrisi karbohidrat yang tinggi, lalu daunnya dapat dijadikan sayuran. Di Indonesia, singkong adalah hasil produksi pertanian ke dua terbesar sesudah padi, sehingga tnaman singkong memiliki potensi sebagai bahan baku yang penting untuk banyak produk makanan dan industri. Untuk citra daun sehat pada tanaman singkong, dapat ditunjukkan seperti pada Gambar 2.1.

Gambar 2.1 Daun singkong sehat

Gambar 2.1 merupakan gambar daun singkong yang sehat. Ciri dari daun singkong yang sehat memiliki warna hijau muda atau hijau tua yang tampak segar. Berikut merupakan penyakit daun singkong yang digunakan.

1. CMD

CMD disebabkan oleh virus yang termasuk dalam *genus Begomovirus* dalam *famili Geminiviridae*. Gejalanya meliputi distorsi lamina daun, mosaik klorosis, bintik-bintik dan pengurangan ukuran tanaman dibandingkan dengan daun yang sehat. Di antara gejala yang biasa terlihat di lapangan adalah pola mosaik pada daun, yang warnanya dapat berkisar dari hijau pucat hingga kuning keputihan. Tingkat klorosis pada permukaan daun bervariasi antara kurang dari 5% sampai hampir 100%. Untuk penyakit CMD pada daun singkong ditunjukkan seperti pada Gambar 2.2.

Gambar 2.2 Daun Singkong dengan penyakit CMD

2. CBSD

CBSD adalah penyakit virus singkong yang cukup berbahaya. Penyakit ini ditandai dengan gejala guratan dengan warna kekuningan dan berbintik-bintik serta

nekrosis pada akar penyimpanan. Nekrosis adalah kondisi saat sel dalam tubuh mengalami cedera dan akan mengakibatkan matinya sel dan juga jaringan tubuh. Untuk penyakit CBSD pada daun singkong ditunjukkan seperti pada Gambar 2.3.

Gambar 2.3 Daun singkong dengan penyakit CBSD

3. *Cassava Bacterial Blight* (CBB)

Penyakit CBB disebabkan oleh *xanthomonas campestris*. Penyakit ini tersebar di negara-negara produsen ubi kayu seperti Amerika Selatan, Afrika dan Asia. Pada Indonesia saat ini, penyakit CBB telah diketahui terdapat di Jawa, Sumatera, dan Kalimantan Selatan. Gejala awal berupa bintik-bintik bersudut dan basah pada daun. Bintik-bintik berkembang dengan cepat terutama di sepanjang tepi daun, dan berubah menjadi coklat dengan tepi kuning. Hal ini mengaikabatkan tanaman menjadi layu dan mati. Untuk penyakit CBB pada daun singkong, dapat ditunjukkan seperti pada Gambar 2.4.

Gambar 2.4 Daun singkong dengan penyakit CBB

4. *Cassava Green Mite* (CGM)

Penyakit ini menyebabkan bercak putih pada daun singkong. Bercak menyebar dari sedikit sampai menutupi seluruh daun sehingga menyebabkan hilangnya klorofil. Daun yang rusak akibat CGM juga dapat menunjukkan gejala berbintik-bintik yang mungkin dapat disalah artikan dengan gejala penyakit *cassava mosaic disease* (CMD). Daun yang sudah rusak parah karena CGM akan menyusut, mengering, dan rontok. Gambar CGM dapat ditunjukkan seperti pada Gambar 2.5.

Gambar 2.5 Daun singkong dengan penyakit CGM

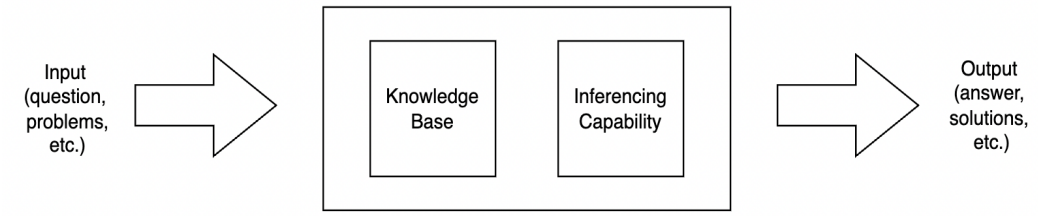
2.3 *Artificial intelligence*

Artificial Intelligence atau kecerdasan buatan sesungguhnya telah diawali semenjak tahun 1956. Pada masa itu segerombol ahli komputer, ahli serta periset dari cabang ilmu lain dari bermacam perguruan tinggi, pabrik dan bermacam golongan terkumpul di Dartmouth College untuk mangulas kemampuan komputer dalam bentuk mengikuti ataupun mensimulasi keahlian orang. Sebagian akademikus yang ikut serta merupakan Allen Newel, Herbert Simon, Marvin Miskey, Oliver Selfridge, serta John McCarthy. Semenjak kata itu, para pakar mulai bekerja *keras* untuk membuat, membahas, mengubah serta meningkatkan hingga menggapai titik perkembangan yang penuh. Mulai dari laboratorium hingga pada lapangan.

Menurut Goralski dan Tan, 2020, Kecerdasan buatan ataupun *artificial intelligence* dikala ini sudah membuka suatu kejadian terkini di aspek korporasi bidang usaha serta pula pemerintahan. Kecerdasan buatan biasanya berhubungan dengan suatu perlengkapan tolong untuk menciptakan suatu kasus serta

menuntaskan kasus yang lingkungan pada beraneka kasus di aspek bidang usaha, korporasi, serta pula pemerintahan. Rancangan penting dari kecerdasan buatan merupakan hasil suatu perlengkapan mesin yang bisa berasumsi seperti manusia (Goralski & Tan, 2020).

Menurut John Mc Carthy, 1956, *Artificial intelligence* dipakai untuk mengidentifikasi serta memodelkan proses–proses pemikiran manusia serta merancang mesin agar bisa mengikuti sikap manusia pada umumnya. Kecerdasan buatan dapat menjadi pilihan untuk mengurangi tingkat pengambilan keputusan yang bersumber pada perasaan. Pada penelitian Bullock (2019), dilakukan suatu perbandingan antara manusia dan kecerdasan buatan dalam menangani suatu permasalahan. Hasilnya, kecerdasan buatan memimpin dalam kasus yang memerlukan suatu kemampuan analisa yang tinggi dengan batas tingkat ketidakpastian dan kesulitan yang rendah. Terbebas dari perihal itu, metode *deep learning* dan *artificial intelligence* akan diperkirakan juga bisa menyelesaikan kasus yang memiliki tingkat ketidakpastian yang tinggi. (Bullock, J. B., 2019).



Gambar 2.6 *Artificial intelligence* pada komputer

Pada Gambar 2.6, *Artificial intelligence* pada komputer membutuhkan sebuah *input,* karena komputer tidak mungkin memperoleh pengetahuannya sendiri dengan belajar, pengalaman atau melakukan riset, tetapi ia memperolehnya melalui usaha yang diberikan oleh seorang manusia. Hampir semua basis pengetahuan (*knowledge base*) itu terbatas, karena difokuskan pada masalah yang khusus. Untuk itu, saat basis pengetahuan itu sudah terbentuk, teknik kecerdasan buatan dapat dipakai untuk memberi keahlian baru kepada komputer supaya dapat berfikir, menalar, dan membuat inferensi (mengambil keputusan berdasarkan pengalaman) serta membuat pertimbangan-pertimbangan yang didasarkan kepada kenyataan dan

hubungan yang ada dalam basis pengetahuan itu. Dengan keahlian - keahlian tersebut, komputer dapat digunakan sebagai alat bantu yang efisien dalam memecahkan kasus dan pengambilan keputusan.

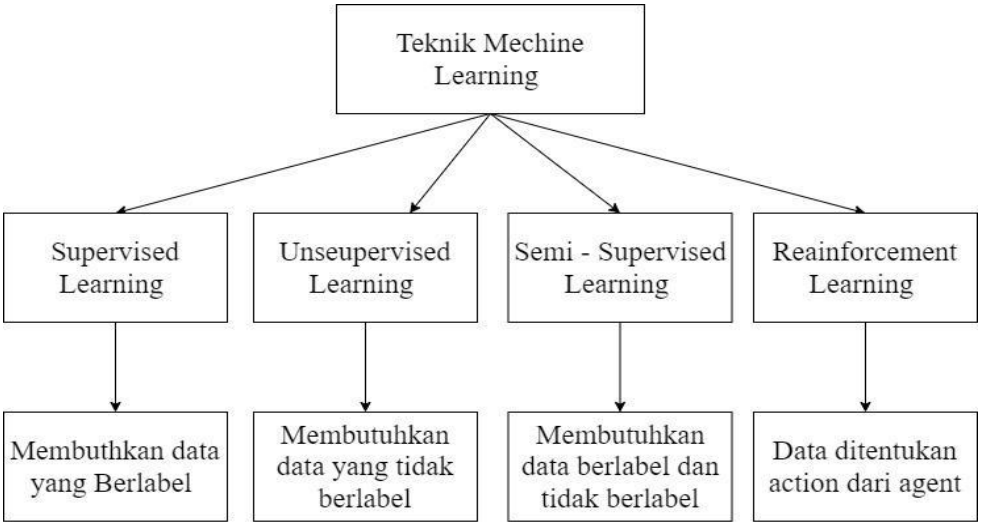
2.4 *Machine Learning*

Machine Learning adalah cabang pada *Artificial intelligence* yang sering dipakai untuk meniru perilaku manusia untuk menyelesaikan kasus atau melakukan automatisasi. *Machine Learning* mencoba mempelajari bagaimana manusia belajar dan mengeneralisasi. Terdapat dua aplikasi utama dalam *Machine Learning* yaitu, klasifikasi dan prediksi. Karakteristik dari *Machine Learning* adalah adanya proses pelatihan, pembelajaran, dan pengujian. Oleh sebab itu, *Machine Learning*, memerlukan data untuk dipelajari yang disebut sebagai data *training*. Klasifikasi merukapan tata cara pada *Machine Learning* yang dipakai oleh mesin untuk memilah atau mengelompokkan obyek berdasarkan karakteristik khusus sebagaimana manusia berupaya membedakan barang satu dengan lainnya (Ahmad, 2017).

Perbedaan antara mesin dan manusia adalah kecerdasan yang manusia bisa belajar dari pengalaman yang sudah dilewati dengan menganalisis informasi dan membuat keputusan. sedangkan mesin tidak mempunyai kecerdasan yang dapat mencari pengalaman sendiri. Namun, mesin mempunyai yang namanya kecerdasan buatan, sehingga mesin dapat melakukan deprogram untuk menganalisis dan mengambil keputusan seperti manusia. *Machine learning* dimaksudkan untuk membatu pekerjaan manusia agar lebih cepat dan lebih tepat dalam menuntaskan kasus, serta bisa mempelajari data melalui kecerdasan buatan (Samsudiney, 2019). *Machine learning* mempunyai empat metode pembelajaran yaitu *supervised learning, unsuervised learning, semi-supervised learning, dan reinforcement learning* (Makovskaja, 2018).

Pada Gambar 2.7, *supervised learning* dirancang untuk melatih algoritma pada kumpulan data berlabel. Hasilnya adalah model pembelajaran yang bisa menuntaskan kasus berdasarkan data *training* yang berlabel. Selain itu, *unsupervised Learning* dirancang untuk melatih algoritma pada data yang tidak berlabel. Model pembelajarannya bertujuan membuat kelompok berdasarkan persamaan pada data *training*. *Semi-supervised learning,* metode ini

mengkombinasikan algortima *supervised* dan *unsupervised,* sehingga data yang diproses terdiri dari data berlabel dan tidak berlabel. Sehingga menghasilkan model yang akurat berdasarkan semua masukan yang telah diberikan. *Reinforcement learning* adalah metode digunakan untuk melatih algoritma berdasarkan konsep "*reward*" dan "*punishment*", model pembelajaran ini dimaksudkan untuk mendapatkan "*reward"* sebesar-besanya dan "*punishment"* sekecil mungkin.

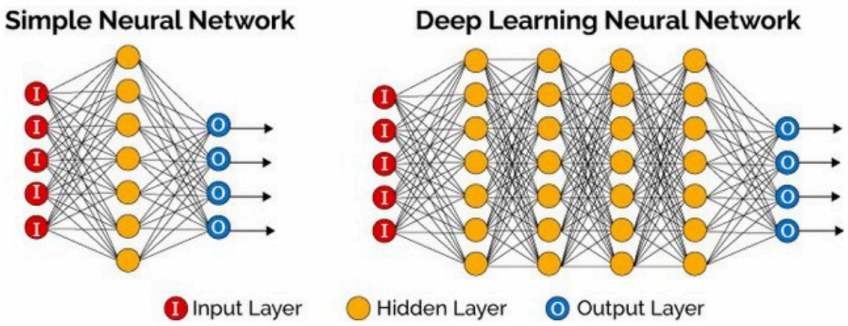


Gambar 2.7 Teknik Pembelajaran mesin (Mitchell, 2006)

2.5 *Deep Learning*

Deep Learning adalah salah satu cabang pembelajaran dari *machine learning*. Bentuk *deep learning* dapat menekuni komputasinya sendiri dengan memakai otaknya sendiri. Deep learning dikonsep untuk terus menganalisa data seperti pada otak manusia dalam mengambil keputusan. Supaya keahlian deep learning semakin mempuni maka deep learning memakai algoritma artificial neural network (Rusydi, 2019).

Algoritma *Deep Learning* yang ditanam pada sebuah komputer dapat belajar mengelompokkan secara langsung dari data berupa gambar, suara, teks, atau video (LeCun, dkk 2015). Dalam implementasinya, *Deep Learning* menggunakan struktur algoritma jaringan saraf tiruan atau *Neural Network.*



Gambar 2.8 Ilustrasi arsitektur *Deep Learning* (Gill, 2020)

Pada Gambar 2.8, *Deep Learning* mempunyai jumlah *hidden layer* yang lebih banyak dibandingkan *Neural Network* biasa. Deep learning mengoptimalkan *hierarchical architecture* didalam mempelajari dan mengerti lingkungan, dikarenakan metode ini mengambil informasi dari data secara mendalam sehingga data tersebut tidak perlu dibagi-bagi menjadi beberapa bagian. Sistemnya memisahkan dan dapat mengetahui nama data terbaik hingga terburuk untuk dijadikan bahan belajar, hal seperti ini dinamakan *high-level feature*. *High-level feature* sendiri adalah fitur yang ada di pemrosesan namun tidak dapat dilihat oleh manusia.

2.6 Pengolahan Citra Digital

Pengolahan citra digital merupakan sebuah ilmu yang mempelajari tentang teknik mengolah citra. Citra yang dibahas disini adalah gambar diam atau foto maupun gambar bergerak. Sedangkan digital disini memiliki arti pengolahan citra yang dilakukan menggunakan komputer. Secara matematis, citra adalah fungsi kontinyu dengan intensitas cahaya pada bidang dua dimensi. Supaya bisa diproses dengan komputer digital, maka suatu citra harus dipresentasikan numerik dengan nilai diskrit.

Citra digital bisa diartikan sebagai sebuah matriks dua dimensi f(x,y) yang berisi kolom M dan Baris N, yang mana pemotongan antara baris dan kolom disebut piksel (*picture element)* atau elemen paling kecil dari sebuah citra (Kusmanto dan Tompunu, 2011). Piksel mempunyai dua parameter, koordinat dan itensitas. Nilai yang ada pada sumbu (x,y) adalah f(x,y), yaitu besar itensitas dari piksel pada titik khusus. Sebuah citra dapat dituliskan pada sebuah matriks :

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & f(0,2) & \cdots & f(0,M-1) \\ f(1,0) & f(1,1) & f(1,2) & \cdots & f(1,M-1) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f(N-1,0) & f(N-1,1) & f(N-1,2) & \ldots & f(N-1,M-1) \end{bmatrix} \quad (1)$$

Berdasarkan persamaan 1., suatu citra f(x,y) dapat dituliskan ke dalam fungsi matematis seperti berikut :

$$\begin{gathered} 0 \leq x \leq M-1 \\ 0 \leq x \leq N-1 \\ 0 \leq f\ (x,y) \leq G-1 \end{gathered} \quad (2)$$

M = Jumlah piksel baris pada array

N = Jumlah kolom pixel pada array

G = Nilai skala keabuan (*grayscale)*

Nilai M,N dan G adalah dipangkatkan dua, sebagaimana terlihat pada rumus 3. :

$$\mathrm{M} = 2\mathrm{m};\ \mathrm{N} = 2\mathrm{n};\ \mathrm{G} = 2\mathrm{k} \quad (3)$$

Nilai m,n dan k merupakan bilangan positif. Interval (0,G) disebut dangan (*grayscale)* . Nilai G tergantung pada proses digitalisasinya, biasanya nilai keabuan 0 menyatakan itensitas hitam dan 1 menyatakan itensitas putih.

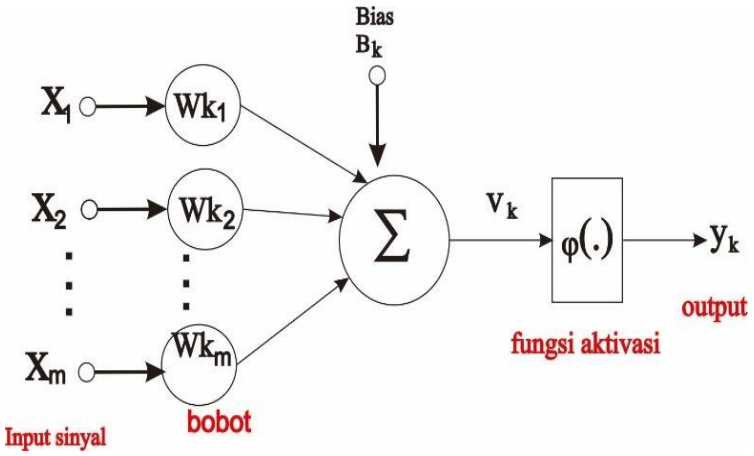
2.7 *Artificial Neural Network*

Kecerdasan buatan atau *artificial intelligence* merupakan sebuah studi tentang bagaimana membuat komputer dapat melakukan pekerjaan manusia. salah satu wujud dari *artificial intelligence* yakni *artificial neural network* atau biasa dibesbut jaringan syaraf tiruan. *Artificial neural network* ini memakai cara kerja otak manusia dengan tujuan mencontoh kecerdasan yang dipunyai oleh manusia. Otak manusia memproses informasi yang diperoleh dari alat indera. Pemrosesan dilakukan oleh neuron yang saling terhubungan antar jaringan syaraf dan bekerja pada sinyal listrik yang melewatinya. Jaringan syaraf menggunakan logika buka

tutup gerbang untuk mengalamatkan sinyal listrik sesuai dengan objek yang diperoleh dari alat indera (Giuseppe Ciaburro, 2017).

Artificial neural network mempunyai langkah kerja seperti jaringan syaraf manusia, sehingga dapat dikatakan bahwa *Artificial neural network* adalah suatu generalisasi matematis dari cara kerja jaringan syaraf manusia. (Giuseppe Ciaburro, 2017) memberikan hipotesis cara kerja dari *Neural Networks* sebagai berikut:

1. Pemrosesan informasi terjadi pada neuron.
2. Sinyal mengalis diantara sel syaraf melalui suatu node.
3. Setiap node memiliki bobot yang bersesuaian. Bobot ini dipakai untuk menduplikat sinyal yang dikirim oleh neuron sebelumnya.
4. Setiap neuron menggunakan fungsi aktivasi pada hasil penjumlahan sinyal dengan bobot untuk menentukan keluaran sinyal dari neuron dan disalurkan ke *node* dan neuron selanjutnya.



Gambar 2.9 Arsitektur Jaringan Saraf Tiruan (Siang, 2004)

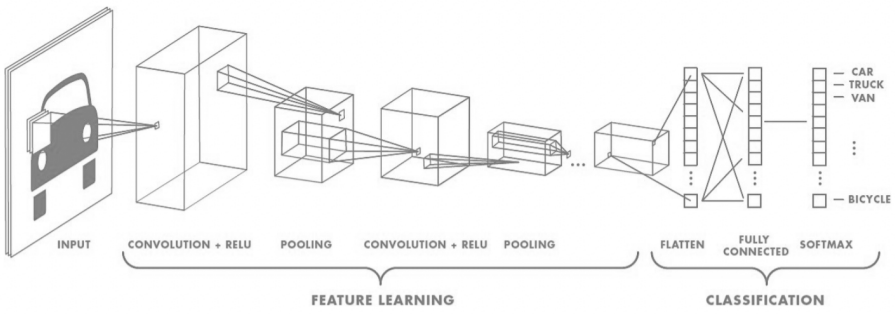
Pada Gambar 2.9, x merupakan neuron, W merupakan bobot dan Σ merupakan Bias. Setelah bias akan diberikan fungsi aktivasi dan setelah itu menghasilkan *Output* yaitu Y. pabila nilai fungsi aktivasi lumayan kuat, maka sinyal akan dilanjutkan. Nilai fungsi aktivasi (*Output* model jaringan) juga bisa digunakan sebagai dasar untuk merubah bobot (Siang, 2004).

2.8 *Convolutional Neural Network* (CNN)

CNN adalah arsitektur yang sanggup mengidentifikasi data prediktif suatu subjek seperti gambar, teks, suara, dan lain - lain. CNN adalah pengembangan dari *multilayer perceptron* yang didesain untuk mengolah data dalam bentuk citra. CNN tercantum dalam tipe *Deep neural network*, karena kedalaman jaringannya dan banyak diterapkan pada data citra. Riset mengenai CNN pertama kali dilakukan oleh Hubel dan Wiesel (1968) mengenai visual cortex pada indra penglihatan kucing.

Secara teori bisa dibilang kalau CNN adalah wujud pengembangan terbaru dari jaringan syaraf tiruan pada aplikasi pengolahan citra yang direpresentasikan pada suatu matriks. Pengembangan yang dimaksud yakni kenaikan lapisan konvolusi dan lapisan pooling saat sebelum data masukan diproses melalui lapisan tersembunyi yang saling tersambung (*fully connected*), setelah itu dilanjutkan ke lapisan luar.

Arsitektur CNN terdiri atas beberapa layer yaitu *convolution, activation function, pooling, dan fully connected.* Ilustrasi jaringan arsitektur Convolutional Neural Network dapat dilihat pada Gambar 2.10.



Gambar 2.10 Arsitektur CNN (Saha Sumit, 2018)

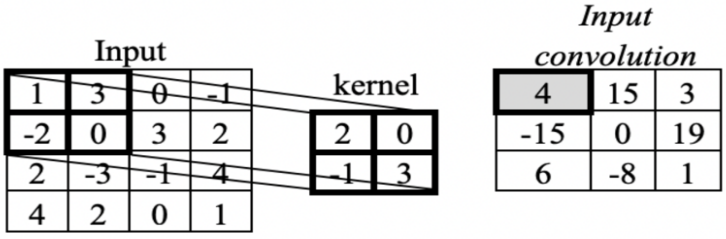
Tahap pertama dalam arsitektur CNN adalah tahap konvolusi. Kemudian dilanjutkan fungsi aktivasi menggunakan ReLu (*Rectifier Linear Unit*), kemudian dilanjutkan dengan proses *pooling*. Proses ini akan diulangi terus - menerus hingga mendapatkan peta fitur yang bagus untuk dilanjutkan ke *fully connected layer*, sehingga dapat diklasifikasikan dan menghasilkan *output class*.

2.8.1 *Convolution Layer*

Convolution layer adalah proses utama yang melandasi jaringan arsitektur CNN dan terdiri atas beberapa *kernel*. *Kernel-kernel* pada lapisan ini disebut sebagai filter konvolusi. *Kernel* berperan mempelajari fitur - fitur lokal pada *feature map*. *Convolutional layer* melaksanakan proses konvolusi pada *output* dari lapisan sebelumnya. Konvolusi merupakan sebutan matematis pada pengaplikasian suatu fungsi pada *output* untuk fungsi lain secara berulang. Konvolusi 2 buah fungsi f(x) dan g(x) didefinisikan sebagai berikut :

$$h(x)=f(x)*g(x)=\int \infty f(a)g(x-a)da \quad (4)$$

Tujuan proses konvolusi pada citra yakni untuk mengekstraksi fitur dari citra masukan. Konvolusi akan menciptakan transformasi linear dari data masukan sesuai informasi spasial pada data. Bobot pada lapisan tersebut menspesifikasikan *kernel* konvolusi yang digunakan, sehingga *kernel* konvolusi bisa dilatih berdasarkan masukan pada CNN (Suartika dkk, 2016). Contoh ilustrasi pada proses *convolution layer* dengan *stride 2* dapat dilihat pada Gambar 2.11.



Gambar 2.11 Ilustrasi Proses *Convolutional Layer* (Khan dkk., 2018)

Adapun perhitungan nilai pada *input convolution* yang didapat adalah:

$$h1,1 = (1.2) + (3.0) + (-2.-1) + (0.3) = 4$$
$$h1,2 = (3.2) + (0.0) + (0.-1) + (3.3) = 15$$
$$h1,3 = (0.2) + (-1.0) + (3.-1) + (2.3) = 3$$
$$h2,1 = (-2.2) + (0.0) + (2.-1) + (-3.3) = -15$$
$$h2,2 = (0.2) + (3.0) + (-3.-1) + (-1.3) = 0$$

$$h2,3 = (3.2) + (2.0) + (-1.-1) + (4.3) = 19$$
$$h3,1 = (2.2) + (-3.0) + (4.-1) + (2.3) = 6$$
$$h3,2 = (-3.2) + (-1.0) + (2.-1) + (0.3) = -8$$
$$h3,3 = (-1.2) + (4.0) + (0.-1) + (1.3) = 1$$

Setelah dihitung, didapatkan matriks *input convolution* :

Tabel 2.1 Matriks *input convolution*

4	15	3
-15	0	19
6	-8	1

2.8.2 *Activation Function*

Activation function adalah sebuah node yang ditambahkan di akhir keluaran dari setiap jaringan syaraf. Pada arsitektur CNN, fungsi aktivasi dapat digunakan setelah proses konvolusi maupun pooling layer. Fungsi aktivasi memiliki banyak jenis, namun yang sering digunakan dalam pemrosesan yaitu *non-linear functions,* karena memungkinkan model melakukan operasi dalam penanganan yang kompleks pada input dan *output* suatu jaringan yang besar (Kapkar, 2020). Berikut jenis fungsi aktivasi *non-linear functions* diantaranya (Nwankpa dkk, 2018) :

1. *Rectified Linear Unit* (ReLu)

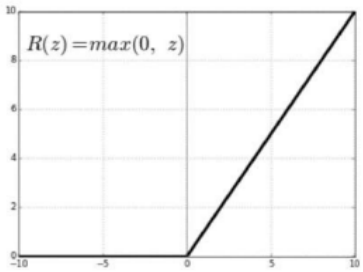
ReLu merupakan fungsi aktivasi *non-linear* yang akan mengkonversi nilai masukan kurang dari 0 (nol) menjadi nilai 0 (nol), dan akan mengeluarkan nilai sama dengan masukkan jika nilai masukkan lebih dari 0 (nol). Fungsi aktivasi ReLU membuat seluruh nilai piksel yang negatif pada suatu citra menjadi nol dan mempertahankan nilai yang positif. Diantaranya kelebihan dari fungsi aktivasi ReLU (Maulana, 2020):

a) ReLu merupakan fungsi aktivasi *default* ketika mengembangkan *Multi-Layer Perceptron* dan CNN.
b) Mengatasi masalah *Gradient Dencent* yang hilang, yang memungkinkan model jaringan dapat belajar lebih cepat dan memiliki kinerja lebih baik.

c) Men-*training* jaringan lebih cepat sehingga dapat mengurangi kemungkinan terjadi *Overfitting*.

Representasi fungsi aktifasi ReLu dituliskan dalam persamaan:

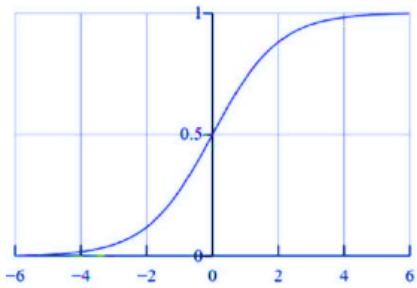
$$ReLu(a) = \max(0, a) \tag{5}$$



Gambar 2.12 Grafik Fungsi *ReLu* (Nwankpa dkk, 2018)

2. *Sigmoid*

Sigmoid merupakan salah satu fungsi aktivasi yang dipakai untuk pembelajaran mesin paling utama pada aspek *logistic regression,* peranan aktivasi ini kerap dipakai untuk identifikasi pembelajaran mesin, akan tetapi tidak sering digunakan untuk model pembelajaran mesin pada tingkat lanjut (Fachrurrozi, 2021). Lalu fungsi aktivasi sigmoid ini biasanya juga digunakan untuk klasifikasi dari dua kelas atau kelompok data. Illustrasi Grafik *Sigmoid* dapat dilihat pada Gambar 2.13.



Gambar 2.13 Grafik Fungsi *Sigmoid* (Nwankpa dkk, 2018)

Lalu berikut ini rumus persamaan dari fungsi aktivasi sigmoid :

$$f(x) = \frac{1}{1+e^{-x}} \tag{6}$$

3. *Softmax*

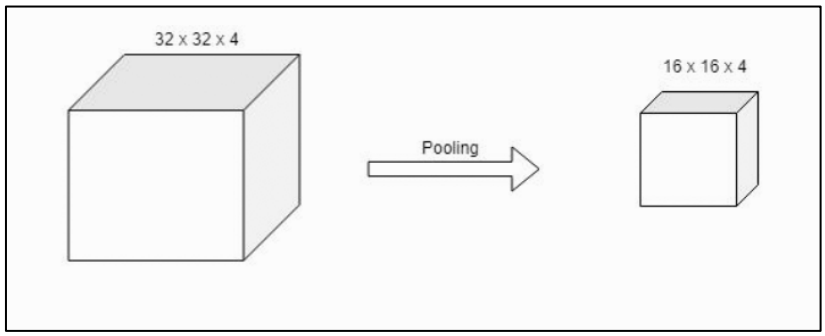
Fungsi *Softmax* merupakan fungsi yang mengganti nilai riil K menjadi vector nilai rill K yang berjumlah 1. Nilai masukan dapat berupa nilai positif, negatif, nol, atau lebih besar dari satu. Namun fungsi *Softmax* mengubahnya jadi angka antara 0 dan 1, alhasil bisa dimaksud sebagai probabilitas. Apabila salah satu masukan kecil (negatif) fungsi *Softmax* mengubahnya jadi probabilitas kecil dan jika masukan besar (positif) fungsi *Softmax* mengubahnya menjadi probabilitas besar, namun nilai bakal selalu antara 0 dan 1 (Wood, 2019). Berikut persamaan dari fungsi softmax :

$$f_j(z) = \frac{e^{z_j}}{\sum k^{e^{z_k}}} \tag{7}$$

Notasi f_j menunjukkan hasil fungsi untuk setiap elemen ke-j pada vektor keluaran kelas. Parameter z merupakan hipotesis yang diberikan oleh model pelatihan agar bisa diklasifikasi oleh fungsi softmax. *Softmax function* memberikan hasil yang lebih impulsif serta mempunyai interpretasi probabilistik yang lebih baik daripada algoritma klasifikasi lainnya. *Softmax* membolehkan kita untuk menghitung probabilitas setiap label. Pada label akan diambil sebuah vektor yang nilainya riil dan mengubahnya menjadi vektor yang bernilai antara nol dan satu, yang apabila semua dijumlahkan maka bernilai satu.

2.8.3 *Pooling Layer*

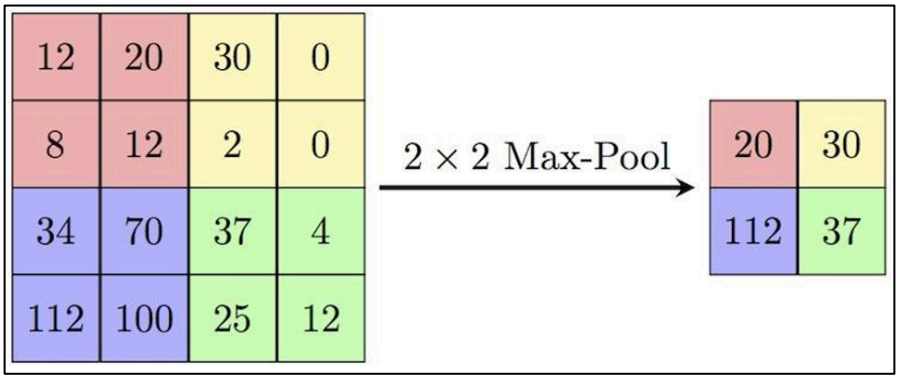
Citra yang sudah melalui konvolusi akan dimasukkan ke dalam *pooling layer*. *Pooling layer* berperan untuk menyederhanakan ukuran representasi spasial dan juga untuk mengurangi jumlah parameter dan komputasi pada jaringan serta menjauhi terbentuknya *overfitting*. Metode pengerjaan dari *pooling layer* adalah dengan menjalankan operasi pada masing-masing kedalaman bagian input dan untuk mengurangi ukuran. Ilustrasi untuk proses *pooling* dapat dilihat pada Gambar 2.14.



Gambar 2.14 Ilustrasi *Pooling* (Putra, 2020)

1. *Max pooling*

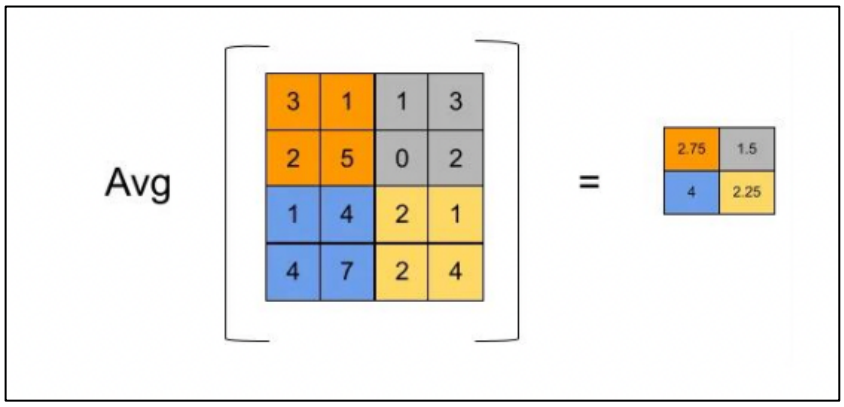
Max Pooling adalah proses konvolusi di mana *Kernel* mengekstrak nilai maksimum dari area yang dililitnya. *Max Pooling* hanya akan meneruskan informasi, jika itu adalah informasi amplitudo terbesar yang tersedia. *Max-pooling* menjaga angka piksel yang terbesar dari masing - masing input, dengan dimensi matriks m x n (Mishkin, dkk, 2016).



Gambar 2.15 Ilustrasi *Max Pooling* (Putra, 2020)

2. *Average Pooling*

Average pooling tidak terlalu sering digunakan dikarenakan alih-alih mengambil nilai jumlah maksimum dalam setiap filter, tetapi mengambil nilai rata-rata dari setiap filter.



Gambar 2.16 *Avarege Pooling*

Pada Gambar 2.16, rata-rata nilai dalam warna oranye adalah 2,75. Rata-rata tersebut didapatkan dari filter *hyperparameter* yaitu *stride 2*.

2.8.4 *Batch normalization*

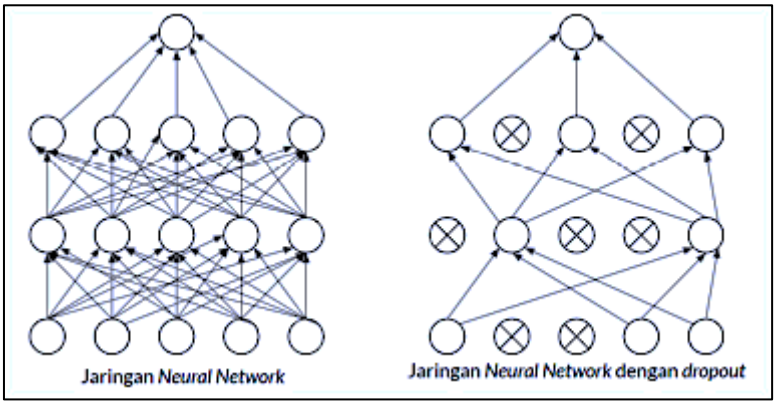
Normalisasi adalah mengubah nilai numerik dalam *dataset* untuk dijadikan skala tertentu tanpa menghapus informasi yang ada di *dataset* tersebut (Singla, 2020). Namun di *machine learning*, teknik dalam melakukan normalisasi dari inputan disebut *batch normalization* (Peccia, 2011). Untuk meningkatkan stabilitas proses pembelajaran pada model, maka dapat dilakukan *batch normalitazion*. *Batch normalization* melakukan pengurangan setiap nilai rata-rata dari *batch*, lalu dibagi dengan nilai standar deviasi dari *batch* tersebut.

2.8.5 *Dropout Layer*

Dropout Layer adalah lapisan yang dapat digunakan untuk mencegah *overfitting* serta mempercepat proses pelatihan. *Dropout* membuang neuron yang berupa *hidden layer* ataupun *layer* yang terlihat pada jaringan. Neuron yang hilang akan dipilih secara *random* oleh sistem dengan peluang dari nol hingga satu. Gambaran dari proses dropout dapat dilihat pada Gambar 2.17.

Pada Gambar 2.17, yang bagian kiri merupakan jaringan saraf tiruan tanpa menggunakan *dropout* yang memiliki dua lapisan tersembunyi. Sedangkan pada bagian kanan merupakan jaringan saraf tiruan menggunakan *dropout*. Dari gambar tersebut tampak bahwa jaringan saraf tiruan yang menggunakan *dropout* terdapat beberapa *neuron* aktivasi yang tidak digunakan lagi. Penggunaan teknik ini pada

model CNN akan berdampak pada performa model dalam melakukan pelatihan serta mengurangi terjadinya *overfitting* (Srivastava dkk, 2014).

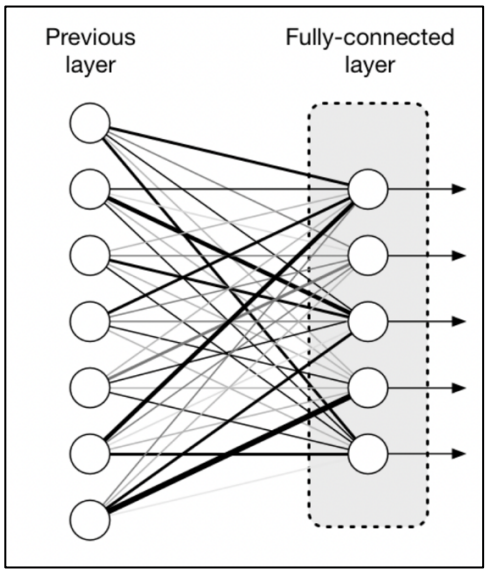


Gambar 2.17 Ilustrasi *dropout layer* (Srivastava dkk, 2014)

2.8.6 *Fully Connected Layer*

Citra yang sudah melewati lapisan konvolusi dan lapisan pooling akan diproses pada *fully connected layer*. *Fully Connected Layer* adalah lapisan di mana semua neuron aktivasi dari lapisan sebelumnya terhubung semua dengan neuron di lapisan selanjutnya seperti halnya jaringan syaraf tiruan biasa. Setiap aktivasi dari lapisan sebelumnya harus diubah menjadi data satu dimensi telebih dahulu, sebelum bisa disambungkan ke semua neuron pada lapisan *Fully-Connected*. Lapisan *Fully-Connected* umumnya digunakan pada metode *Multi Lapisan Perceptron* yang bertujuan untuk mengolah data sehingga bisa diklasifikasikan.

Perbedaan lapisan *Fully-Connected* dan lapisan konvolusi biasa yakni pada lapisan konvolusi biasa, neuron terhubung hanya ke tempat tertentu pada masukan, sementara lapisan *Fully-Connected* mempunyai neuron yang secara keseluruhan terhubung. Namun, kedua lapisan tersebut masih menggunakan *dot product*, sehingga fungsinya tidak terlalu berbeda. *Fully Connected Layer* pada intinya adalah sebuah *neural network multilayer perceptron*, yang memiliki beberapa *hidden layer*, *activation function*, *output* layer dan *loss function*. *Fully connected layer* yang akan berperan untuk mengklasifikasi data masukan. Ilustrasi untuk *fully connected layer* dapat dilihat pada Gambar 2.18.



Gambar 2.18 *Fully Connected Layer*

Lalu bentuk persamaan *hidden layer* dan *output* layer pada *fully connected layer* adalah sebagai berikut :

$$Z = b + \sum_{i=1}^{n} x_i w_i \tag{8}$$

w_i adalah *weight* yang ada pada tiap iterasi, x_i adalah input, n adalah jumlah input yang terdapat pada *layer*, dan b merupakan nilai bias. *Output* yang dihasilkan dari *pooling layer* masih berbentuk *multidimensional array*, sehingga akan di-*flatten* terlebih dahulu untuk mengubah data menjadi vektor sebelum dijadikan input untuk *fully connected layer.*

2.8.7 *Loss Function*

Fungsi dari *loss function* adalah untuk menguantifikasi perbedaan antara hasil yang diharapkan dan hasil yang diberikan oleh model *machine learning*. Berikut adalah *loss funcition* yang sering digunakan :

1. *Regression loss functions*

Salah satu jenis dari fungsi loss ini salah satunya adalah *Mean Square Error* (MSE), keunggulan dari MSE sendiri adalah mudah dipahami, mudah diterapkan.

Untuk menghitung MSE rata-rata selisih kuadrat antara prediksi dan data sebenarnya.

2. *Binary classification loss functions*

Fungsi dari loss functions adalah digunakan untuk masalah klasifikasi dan merupakan salah satu paling popular. Ini hanya mofikasi langsung dari fungsi kemungkinan dengan algoritma. Salah satu jenis dari *loss function* adalah *binary cross entropy.*

3. Multi-class classification loss functions

Jenis ini sering dipakai dalam klasifikasi karena jenis ini mampu digunakan dibanyak label kelas. Mempunyai 2 yang sering dipakai yakni *Categorical Cross Entropy* (CCE) dan *Sparse Categorical Cross Entropy* (SCCE), Perbedaan dari kedua fungsi tersebut adalah bentuk dari hasil keluarannya, CCE menghasilkan label berformat *One-Hot Encoding* (OHE) yang berisi kemingkinan kecocokan untuk setiap kategori, sedangkan fungsi yang satunya SCCE menghasilkan label berformat intiger yang berisi indeks kategori dari kategori yang paling cocok (Pratama, 2022).

2.8.8 *Optimization Function*

Optimatizion function adalah fungsi yang sering digunakan untuk memperbarui parameter weight bertujuan agar meminimalisir nilai loss. Didalam deep learning ada beberapa *optimizer* yang paling sering digunakan, yaitu :

1. *Adaptive Gradient Algorithm* (Adagrad)

Sebuah metode *adaptive learning rate*, parameter untuk metode ini adalah *learning rate*, Adagrad menghilangkan kebutuhan dalam melakukan penyesuaian *learning rate* secara manual. Namun, algoritma optimasi ini memiliki kelemahan yaitu *learning rate* semakin menurun sampai tidak dapat belajar lagi. RMSprop dan adam mencoba untuk menyelesaikan permasalannya.

2. *Root Mean Square Propagation* (RMSProp)

RMSprop mencoba untuk menyelesaikan permasalahan yang ada di adagrad menggunakan *moving average* dari gradien kuadrat. Menggunakan besarnya dari gradien discent untuk menormalkan gradien. *Learning rate* di RMSprop disesuaikan dengan cara otomatis lalu memanfaatkan *learning rate* yang berbeda-beda di setiap parameternya.

3. *Adaptive Movement Estimation* (Adam)

Adam merupakan pengembangan dari Adagrad seperti yang sudah dijelaskan diparagraf sebelumnya. Adam merupakan kombinasi antara RMSProp dan *Stochastic Gradient Descent* (SGD) dengan momentum (Bushaev, 2018). Adam sangat efisien secara komputasi dan memiliki kebutuhan memori yang sedikit. *Adam Optimizer* merupakan salah satu algoritma komputasi gradient discent yang popular.

4. *Stochastic Gradient Descent* (SGD)

Algoritma ini mencari sebuah weight baru dengan melakukan pengambilan salah satu data dari seluruh data *training*. Kuntungan menggunakan SGD itu mampu meminimalisir penggunaan memori yang dibutuhkan saat pemrosesan *weight* baru (Pratama, 2022).

2.9 *Transfer Learning*

Ada dua cara pendekatan dasar untuk melatih model CNN, yakni dengan membentuk jaringan dari awal (*Scratch*) atau dengan pembelajaran transfer (*Transfer Learning*) (Altunta dkk. 2019). Melatih model CNN dari awal membutuhkan banyak sample objek untuk mengungkap fitur deskriptif yang ada dalam citra. Meskipun pendekatan ini memberikan kendali penuh terhadap jaringan yang dibuat, namun waktu yang dibutuhkan untuk melakukan pelatihan model akan sangat lama, selain itu juga masalah *overfitting* dan konvergensi adalah permasalahan yang potensial dihadapi dalam proses pelatihan.

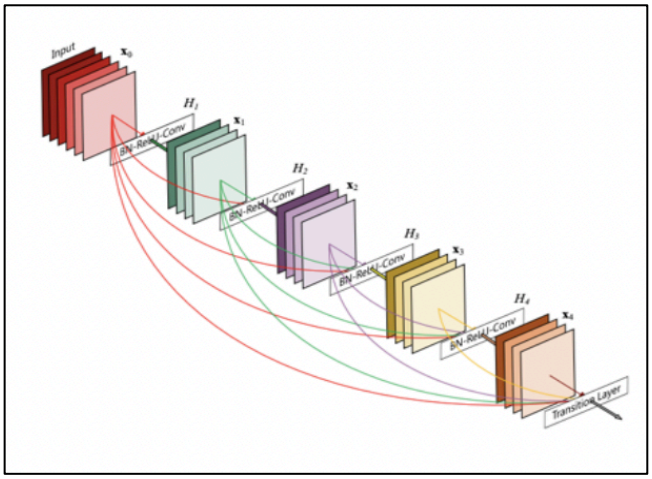
Untuk menangani masalah ini, model CNN dilatih dengan menggunakan *fine-tuning* atau biasa disebut sebagai *transfer learning.* Dalam pendekatan pembelajaran transfer yang umum, lapisan konvolusi digunakan sebagai fitur ekstraktor yang tetap, dan hanya *fully-connected layer* yang disesuaikan dengan tugas klasifikasi yang baru. *Fine-tunning* dimulai dengan men-*transfer* bobot dari jaringan yang telah dilatih ke jaringan baru yang akan dilatih.

2.10 *DenseNet*

DenseNet merupakan jaringan konvolusi yang saling terkoneksi. *DenseNet* memanipulasi jaringan melalui penggunaan kembali fitur, cara ini membutuhkan parameter yang lebih sedikit dibandingkan dengan model CNN lain yang setara. *DenseNet* terdiri dari blok-blok bernama *Dense Blocks*, di mana dimensi dari peta

fitur tetap konstan di dalam blok, tetapi jumlah filter berubah di antara mereka. Dan diantara masing-masing blok terdapat transition layer yang mengurus *down sampling*.

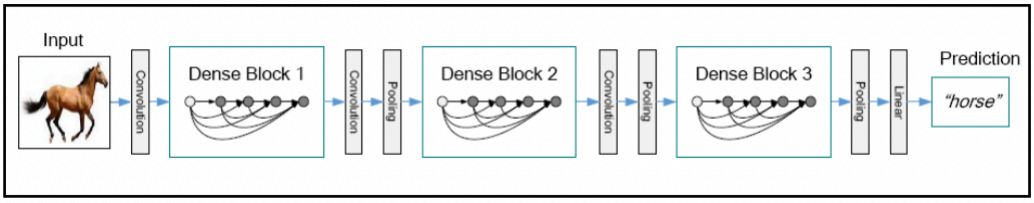
Jaringan pada *DenseNet* memiliki koneksi langsung L(L+1)/2. L adalah jumlah lapisan dalam arsitektur. Untuk setiap lapisan, peta fitur dari semua lapisan sebelumnya digunakan sebagai input, dan peta fitur sendiri digunakan sebagai input ke semua lapisan berikutnya. *DenseNet* memiliki beberapa keunggulan menarik: meringankan masalah gradien-gradien, memperkuat penyebaran fitur, mendorong penggunaan kembali fitur, dan secara substansial mengurangi jumlah parameter. Menurut CVPR (*Computer Vision and Pattern Recognition*) pada tahun 2017, model *DenseNet* mendapatkan penghargaan sebagai makalah terbaik (Huang dkk. 2017).



Gambar 2.19 Arsitektur *DenseNet*
(G. Huang, dkk. *'Densely Connected Convolutional Networks'* 2018)

Pada Gambar 2.19, untuk setiap komposisi lapisan menggunakan *batch normalization*, ReLU *activation* dan *convolution* dengan *filter 3x3*. Pada setiap blok ada masukan berupa matriks sesuai dengan piksel citra kemudian masuk ke proses *batch normalization* untuk mengurangi adanya *overfitting* pada saat proses *training*, ReLU *activation* untuk hanya membuat pembatas pada bilangan nol, artinya apabila $x \leq 0$ maka $x = 0$ dan apabila $x > 0$ maka $x = x$, *convolution* dengan filter 3x3 proses citra matriks yang sudah dilakukan operasi ReLU *activation* akan dikalikan dengan

matriks *convolution* dengan filter 3x3 dan keluaran yang dihasilkan berupa nilai matriks yang sudah di proses sebelumnya.



Gambar 2.20 Ilustrasi *DenseNet* Yang Berisikan 3 Blok
(Huang dkk., 2017)

Pada *DenseNet* yang menggunakan 3 blok. Lapisan yang berada pada *dense block* berisikan *batch normalization*, *ReLU* dan *convolution* dengan filter 3 x 3. Lapisan antara 2 blok yang berdekatan disebut lapisan transisi yang murubah ukuran fitur melalui *convolution* dan *pooling*. Yang diilustrasikan pada Gambar 2.20.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Gambar 2.21 Ilustrasi layer *DenseNet* Yang Berisikan 4 Blok
(Huang dkk., 2017)

Pada *DenseNet* yang menggunakan 4 blok, lapisan yang berada pada *dense block* berisikan *batch normalization,* ReLU *activation* dan *convolution* dengan filter 3x3. Lapisan antara 2 blok yang berdekatan disebut sebagai *transition layer* dan perubahan ukuran fitur melalui *convolution* dan *pooling (average pooling)*. Untuk tahap klasifikasi nya menggunakan *global average pooling* kemudian tahap

selanjutnya *softmax activation.* Pada Gambar 2. 21 terdapat model *DenseNet-121, DenseNet-169, DenseNet-201,* dan *DenseNet-264*. Disini *DenseNet-121* dipilih untuk mengidentifikasi penyakit singkong berdasarkan citra daun. *DenseNet-121* memiliki susunan 5 + (6 + 12 + 24 + 16) * 2 = 121. Nilai 5 berisi *convolution* dan *pooling layer*, *transition layer* dan *classification layer*. Blok dikalikan 2 karena setiap blok memiliki 2 lapisan (1x1 konvolusi dan 3x3 konvolusi).

2.11 *Python*

Python merupakan salah satu contoh bahasa tingkat tinggi. Contoh bahasa tingkat tinggi yang lain adalah *Pascal, C++, Pert, Java*, dan lain-lainnya. Sedangkan bahasa tingkat rendah adalah bahasa mesin atau biasa disebut bahasa *assembly*. Sederhananya, sebuah komputer hanya bisa menjalankan program yang ditulis dalam bentuk bahasa mesin atau bahasa *assembly*. Oleh karena itu, apabila suatu program ditulis dalam bahasa tingkat tinggi, maka program tersebut perlu diproses dahulu sebelum dapat dijalankan pada komputer. Hal ini adalah salah satu kekurangan bahasa tingkat tinggi yang membutuhkan waktu untuk memproses suatu program sebelum program tersebut dapat dijalankan. Meskipun seperti itu, bahasa tingkat tinggi masih memiliki banyak sekali keuntungan.

Bahasa tingkat tinggi mudah dipelajari, mudah ditulis, mudah dibaca, dan tentu saja mudah dicari kesalahannya. Bahasa tingkat tinggi juga mudah diubah portabel untuk disesuaikan dengan mesin yang menjalankannya. Berbeda dengan bahasa mesin yang hanya bisa digunakan untuk mesin tersebut saja. Dengan berbagai kelebihan ini, maka banyak aplikasi ditulis menggunakan bahasa tingkat tinggi. Proses merubah data pada bahasa tingkat tinggi ke tingkat rendah dalam bahasa pemrograman terdapat dua tipe, yakni *interpreter* dan *compiler* (Utami, 2004).

2.12 *Keras*

Keras merupakan *library* jaringan syaraf tiruan *open source* yang ditulis di Bahasa pemrograman python. *Keras* dapat berjalan pada *Tensor Processing Unit* (TPU) atau GPU yang besar, dan model *Keras* dapat diekspor untuk dijalankan pada *browser* ataupun smartphone. Penggunaan *Keras* sebagai *deep learning framework* didasarkan oleh beberapa hal, yakni : (Surya, 2020)

1. *Keras* mempunyai dokumentasi yang cukup jelas sehingga mudah untuk dipelajari dan digunakan.
2. Pemanfaatan *Keras* sebagai *framework* telah terintegrasi dengan *Tensorflow*
3. *Keras* telah mempunyai banyak platform untuk digunakan: Android, iOS, *Python*, dan *Raspberry Pi*.
4. Mendukung berbagai penggunaan GPU.
5. *Keras* telah menjadi bagian dalam ekosistem *deep learning* yang di-*support* banyak perusahaan terkemuka lainnya seperti : *Mircosoft, Google* dan AWS.

2.13 *Tensorflow*

Tensorflow adalah perpustakaan perangkat lunak, yang diciptakan oleh tim *google brain* dalam organisasi penelitian mesin cerdas *Google*, untuk tujuan melakukan pembelajaran mesin dan penelitian jaringan syaraf mendalam. *Tensorflow* kemudian menyatukan aljabar komputasi dengan teknik pengoptimalan kompilasi, sehingga mempermudah banyak penghitungan ekspresi matematis, yang mana akar masalahnya adalah waktu yang diperlukan untuk melakukan perhitungan. Fitur utama *Tensorflow* meliputi (Taufiq, 2018) :

1. *Tensorflow* dapat Mendefinisikan, mengoptimalkan, dan menghitung secara efisien, ekspresi matematis yang memakai *array* multi dimensi (*tensors*).
2. Pemrograman pendukung jaringan syaraf tiruan dan teknik pembelajaran mesin
3. Penggunaan GPU yang lebih transparan, mengoptimalisasi memori yang sama dan data yang digunakan. *Tensorflow* dapat menulis kode yang sama dan menjalankannya di CPU maupn GPU. Lebih khusus lagi, *Tensorflow* dapat mengetahui bagian perhitungan mana yang harus dipindah ke GPU.
4. Skalabilitas komputasi yang tinggi di seluruh mesin dan kumpulan data yang besar

2.14 *Confusion Matriks*

Performa pada model bisa dilihat dari hasil analisis evaluasinya dengan memakai *confusion matrix. Confusion matriks* menggunakan perhitungan nilai *accuration*, *recall, precision* dan *f1 score* dari sebuah model pembelajaran mesin

(Ghoneim, 2019). Pada perhitungan *confusion matriks,* terdapat empat kondisi kasus yang perlu diperhatikan yaitu :

a. *True Positive* (TP) : Kondisi ketika hasil prediksi benar dan realitanya memang benar.
b. *True Negative* (TN) : Kondisi ketika hasil prediksi benar dan realitanya ternyata salah.
c. *False Positive* (FP) : Kondisi ketika hasil prediksi salah dan realitanya ternyata benar.
d. *False Negative* (FN) : Kondisi ketika hasil prediksi salah dan realitanya memang salah.

Nilai yang diperoleh dalam kondisi kasus tersebut digunakan untuk menghitung akurasi dengan rumus berikut :

$$Akurasi = \frac{TP + TN}{TP + FP + FN + TN} \tag{9}$$

Akurasi adalah perbandingan dari data yang telah diprediksi berdasarkan kelas dari keseluruhan data. Untuk menghitung tingkat presisi prediksi bisa memakai rumus berikut :

$$Presisi = \frac{TP}{TP + FP} \tag{10}$$

Presisi merupakan perbandingan data yang telah dikelompokkan sesuai dengan kelas yang telah diprediksi dengan kondisi positif. Lalu *Recall* dapat dihitung memakai rumus berikut :

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

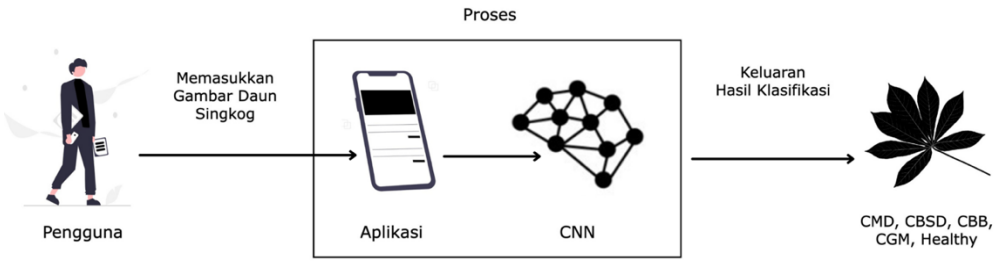
Recall merupakan perbandingan data yang sudah diprediksi berdasarkan kelas dengan data yang sesuai dengan kelasnya. Untuk *F1 Score* dapat dihitung menggunakan rumus berikut :

$$F1 - Score = \frac{2\ X\ (Recall\ X\ Presisi)}{(Recall + Presisi)} \quad (12)$$

F1 Score merupakan perbandingan rata-rata dari nilai presisi dan nilai *recall* yang diperoleh dari proses perhitungan presisi dan *recall*.

2.15 *Model Deployment*

Untuk mengetahui sebuah model telah berfungsi sesuai dengan yang diinginkan, maka diperlukan *deployment model* ke dalam aplikasi, supaya pengguna bisa mengakses dan berinteraksi dengan model yang sudah dibuat (Kawwa, 2019). Berdasarkan hal tersebut, maka model pembelajaran mesin dapat digunakan pada aplikasi mobile.



Gambar 2.22 Alur sistem pendeteksi penyakit singkong berbasis android

Pada Gambar 2.22, Memperlihatkan sebuah sistem pendeteksi penyakit daun singkong sederhana berbasis android. Pada sistem tersebut pengguna dapat mengunggah citra berwarna daun singkong kemudian melakukan klasifikasi penyakit menggunakan model *Transfer Learning* CNN yang telah ditanamkan. Setelah itu aplikasi akan mengeluarkan hasil berupa klasifikasi penyakit singkong.