BAB 2

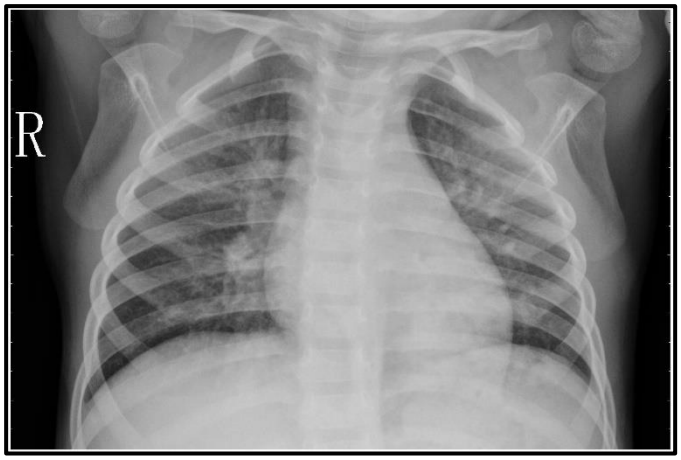
LANDASAN TEORI

2.1 Pneumonia akibat bakteri/virus

Pneumonia adalah penyakit batuk pilek yang disertai dengan napas yang sesak atau napas cepat. Penyakit ini banyak terjadi pada anak balita, namun juga dapat terjadi pada orang dewasa, dan pada orang usia lanjut. Pneumonia atau sering disebut dengan istilah paru paru basah adalah infeksi yang memicu inflamasi pada kantong kantong udara di salah satu atau kedua paru paru. Pada pengidap paru paru basah, sekumpulan kantong kantong udara kecil di ujung saluran pernapasan dalam paru paru akan membengkak dan dipenuhi cairan (Misnadiarly, 2008).

Pneumonia dapat disebabkan oleh berbagai hal, contohnya seperti mikro organisme, namun sebagian besar disebabkan oleh bakteri. Bakteri khas penyebab pneumonia yang paling sering adalah *Streptococcus pneumoniae* yang mengakibatkan sekitar 50% kasus yang ada, *Haemophilus influenza, Klebsiella,* dan *Staphylococcus.* Disebut khas karena beberapa bakteri ini mempunyai kecenderungan menyerang orang yang peka pada penderita pasca infeksi influenza. Bakteri tidak khas yang sering ditemui adalah *Mycoplasma pneumonia, Legionella,* dan *Chlamydia.* Selain bakteri, virus juga merupakan salah satu penyebab pneumonia sebesar 15% dari kasus pneumonia anak dan 30% pneumonia dewasa. Ada juga jamur yang mengakibatkan pneumonia, namun relatif jarang ditemui.

Pneumonia akibat bakteri adalah infeksi paru paru yang umumnya disebabkan oleh *Streptococcus* dan *Haemophilus influenza*. Ketika bakteri tersebut memasuki paru paru, sistem kekebalan tubuh akan berusaha untuk menghancurkan bakteri tersebut. Reaksi kekebalan tubuh ini memicu radang dan penyempitan saluran udara. Apabila ini terjadi, tugas paru paru untuk mengangkut oksigen segar dan mengeluarkan udara kotor akan terganggu. Akibatnya akan timbul berbagai gejala, termasuk sulit bernapas, napas pendek, dan merasa lebih lelah dari biasanya (Lim, 2006).



Gambar 2. 1 X-Ray Pneumonia

(Sumber: https://www.kaggle.com/)

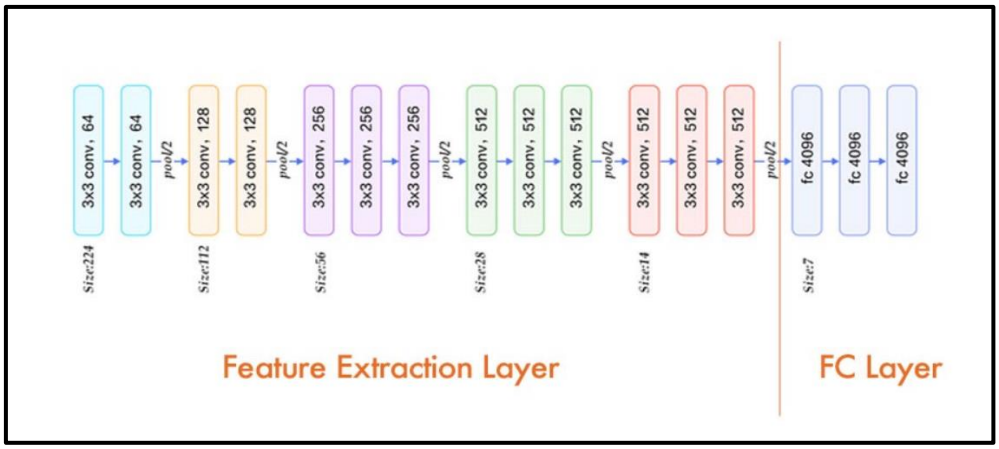
Selain diakibatkan oleh bakteri, penyakit pneumonia juga dapat disebabkan oleh virus. Meskipun gejala yang ditimbulkan sama dengan pneumonia akibat bakteri, namun cara mengobatinya berbeda. Ini karena mikroba yang menyebabkan keduanya pun berbeda. Pneumonia akibat virus menyerang orang orang dengan sistem kekebalan tubuh yang kuat. Sedangkan pneumonia akibat bakteri (Gambar 2.1) biasanya ditemukan pada pasien dengan sistem kekebalan tubuh yang lemah atau baru sembuh dari infeksi pernapasan. Pneumonia akibat bakteri dapat diobati dengan antibiotik, sedangkan pneumonia akibat virus tidak mempan terhadap obat

yang sama. Karena perbedaan ini, penting bagi pasien untuk mendapat diagnosis yang tepat agar dapat diobati dengan baik. Sebagian besar pasien dapat cepat sembuh dengan obat obatan, namun pasien dengan gejala parah atau mengidap gangguan kesehatan perlu diobati di rumah sakit.

2.2 Convolutional Neural Network (CNN) - VGG-16

Convolutional Neural Network (CNN) adalah jenis *machine learning* yang bagus digunakan untuk *visual object recognition* (Boser, 1989). CNN merupakan operasi konvolusi yang menggabungkan beberapa lapisan pemrosesan, menggunakan beberapa elemen yang beroperasi secara paralel dan terinspirasi oleh sistem saraf biologis (Hu et al., 2015). Pada CNN setiap neuron dipresentasikan dalam bentuk 2 dimensi, sehingga metode ini cocok untuk pemrosesan dengan input berupa citra (Maggiori et al., 2016).

Proses ekstraksi dalam CNN terdiri dari beberapa lapisan tersembunyi atau *hidden layer*, yaitu lapisan konvolusi, fungsi aktifasi (ReLU), dan *pooling*. CNN bekerja secara hierarki, sehingga *output* pada lapisan konvolusi pertama digunakan sebagai *input* pada lapisan konvolusi selanjutnya. Pada proses klasifikasi terdiri dari *fully-connected* dan fungsi aktivasi (*softmax*) yang outputnya berupa hasil klasifikasi (Katole et al., 2015).



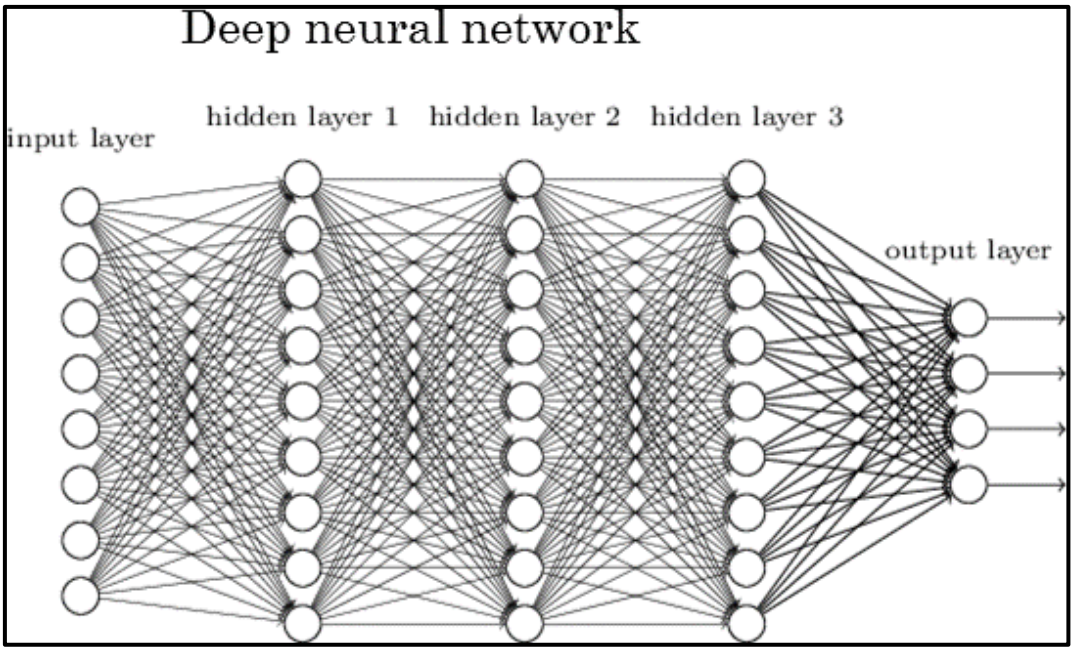
Gambar 2. 2 Arsitektur VGG-16

(Sumber: https://medium.com/)

VGG16 (*Visual Geometry Group*) terdiri dari *16-layer network* yang digunakan oleh Visual Geometry Group di Universitas Oxford. Fitur utama dari arsitektur ini adalah penambahan kedalaman dari *network*. Pada VGG16, gambar 224x224 RGB melalui 5 *blocks convolutional layer* dimana setiap *block* tersusun atas 3x3 *filter*. *Block* dipisahkan oleh *max-pooling layer*. *Max-pooling* dilakukan lebih dari 22 *windows* dengan *stride* 2. 5 *blocks* dari *convolutional layer* diikuti oleh 3 *fully-connected* (FC) *layer* (Gambar 2.). Layer terakhir adalah *soft-max layer* yang menghasilkan *class probabilities* (Simonyan dan Zisserman, 2015).

2.3 Hidden Layer

Hidden layer atau lapisan tersembunyi merupakan layer yang terdiri dari neuron yang menerima data dari *input layer*. Informasi yang diterima pada *input layer* akan dilanjutkan melalui *hidden layer* satu persatu hingga mencapai *output layer*.



Gambar 2. 3 Layer pada Neural Network

(Sumber: https://www.semanticscholar.org/)

Hidden layer berfungsi untuk membantu proses, semakin banyak *hidden layer* yang digunakan maka akan semakin bagus dan semakin cepat didapat *output* yang diinginkan, tetapi waktu *training* akan semakin lama (Mitchell, 1997). Terkadang *machine learning* dapat mengakibatkan *overfitting* dimana *machine learning* terlalu fokus pada *training* dataset tertentu, hingga tidak bisa melakukan prediksi dengan tepat. Beberapa solusi terhadap *overfitting* adalah dengan melaukan validasi pada model, menambahkan jumlah data sampel, ataupun dilakukannya degeneralisasi agar mesin tidak “terlalu rajin” belajar (Christopher, 2016).

2.4 Data Splitter

Dalam membuat model *machine learning*, data harus dibagi menjadi *data train* dan *data test*. Suatu *machine learning* harus memiliki *goal* yang harus dicapai, untuk mencapai *goal* tersebut *machine learning* harus diberitahu *dataset* yang harus dicapai dan *dataset* yang bisa digunakan untuk mencapai *goal* yang diinginkan.

Dataset yang harus dicapai adalah *data test*, sedangkan *dataset* yang digunakan untuk mencapainya adalah *data train*. *Data train* nantinya akan digunakan untuk membuat model *machine learning*, sedangkan *data test* akan digunakan untuk menguji performa dan akurasi dalam model yang dibuat.

Dari seluruh *dataset* yang akan digunakan, akan dibagi untuk menjadi *data train* ataupun *data test*. Pembagian atau perbandingan antara data train dan data test yang paling umum adalah 20% *data test* dan 80% *data train,* terkadang perbandingan yang digunakan bisa 70% dan 30% atau 90% dan 10% (Shulga, 2018).

2.5 Metode Topsis

Metode TOPSIS adalah salah satu metode pengambilan keputusan multikriteria yang pertama kali diperkenalkan oleh Yoon dan Hwang tahun 1981 (Olson, 2006). TOPSIS didasarkan pada konsep dimana alternatif terpilih yang terbaik tidak hanya memiliki jarak terpendek dari solusi ideal positif, namun juga memiliki jarang terpanjang dari solusi ideal negatif dari sudut pandang geometris dengan menggunakan jarak *Euclidean* untuk menentukan kedekatan relatif dari suatu alternatif dengan solusi optimal (Kusumadewi, dkk, 2006).

Solusi ideal positif didefinisikan sebagai jumlah dari seluruh nilai terbaik yang dapat dicapai untuk setiap atribut, sedangkan solusi ideal negatif terdiri dari seluruh nilai terburuk yang dicapai untuk setiap atribut (Meliana, 2011). TOPSIS mempertimbangkan keduanya, jarak terhadap solusi ideal positif dan jarak terhadap solusi ideal negatif dengan mengambil kedekatan relatif terhadap solusi ideal positif. Berdasarkan perbandingan terhadap jarak relatifnya, susunan prioritas alternatif bisa dicapai.

Tahapan dalam metode TOPSIS adalah sebagi berikut.

1. Menentukan kriteria (C_i) dan sifat.
2. Menentukan rating kecocokan setiap alternatif (A_i) pada setiap kriteria (C_i).
3. Membuat matriks keputusan yang ternormalisasi dengan rumus berikut.

$$|x_n| = \sqrt{\sum_{i=1}^{m} x_{ij}^2} \qquad \ldots(2.1)$$

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} x_{ij}^2}} \qquad \ldots(2.2)$$

4. Perkalian antara bobot dengan nilai setiap atribut dengan rumus berikut.

$$y_{ij} = w_i * r_{ij} \qquad \ldots(2.3)$$

5. Menentukan matriks solusi ideal positif dan matriks solusi ideal negatif dengan rumus berikut.

6. Menentukan jarak antara nilai setiap alternatif dengan matriks solusi ideal positif dan negatif dengan rumus berikut.

$$D_i^+ = \sqrt{\sum_{j=1}^{n} \left(y_i^+ - y_{ij}\right)^2} \quad \ldots(2.4)$$

$$D_i^- = \sqrt{\sum_{j=1}^{n} \left(y_{ij} - y_i^-\right)^2} \quad \ldots(2.5)$$

7. Menentukan nilai preferensi untuk setiap alternatif dengan rumus berikut.

$$V_i = \frac{D_i^-}{D_i^- + D_i^+} \quad \ldots(2.6)$$

Nilai V_i yang lebih besar menunjukkan bahwa nilai alternatif A_i adalah yang terbaik. Nilai selanjutnya diurutkan dari yang mendapat nilai terbesar sampai ke yang terkecil sebagai peringkat.

2.6 Fine Tuning

Fine tuning adalah proses yang dipakai untuk melakukan klasifikasi. *Fine tuning* digunakan ketika terdapat sebuah *knowledge* yang didapatkan dari suatu penyelesaian permasalahan dan digunakan untuk permasalahan yang baru namun tetap berhubungan / berkaitan. Jadi, *fine tuning* menggunakan sebuah model yang telah di *train* dari suatu *task* yang kemudian dapat menghasilkan *task* lain yang mirip.

Jika suatu *task* mirip / menyerupai *task* lainnya maka dengan menggunakan *artificial neural network* yang sudah pernah dibuat dan di-*train* dapat membantu untuk membuat model lainnya tanpa harus membuat/*develop* dari awal. Ketika membuat suatu model dari awal, kita harus mencoba dengan banyak pendekatan seperti *trial-and-error* karena kita harus menentukan berapa banyak layer yang

ingin digunakan, tipe seperti apa layer yang ingin digunakan, bagaimana mengurutkan layer, berapa banyak *node* yang dibutuhkan untuk setiap layer, harus menentukan berapa banyak regulasi yang dibutuhkan, *learning rate* apa yang dibutuhkan. Hal tersebut yang membuat metode *fine tuning* sangat membantu jika kita dapat mencari *train model* yang sudah ada terlebih dahulu dan *task* tersebut mirip dengan apa yang ingin diteliti.

Cara menggunakan *fine tuning* untuk mengenali model baru adalah dengan menggunakan model pertama dan menghapus *output layer* karena kita menginginkan *output* dari suatu objek yang baru. Permasalahannya adalah berapa banyak layer yang harus dihapus dan berapa banyak layer juga yang harus ditambahkan untuk mengenali objek baru tersebut karena layer awal dari suatu *network* bisa saja sudah mempelajari tentang fitur yang lebih umum seperti bentuk dan tekstur. Kemudian setelah memodifikasi struktur dari *network* yang ada, kita dapat menyimpan / *freeze original model* yang ada agar *weight* dari layer ini tidak berubah kapanpun dilakukan *train model* pada data yang baru untuk *task* yang baru. Yang dibutuhkan adalah *weight* yang sama setelah di-*train* dari *task* awal, kita hanya menginginkan *weight* pada layer baru untuk di-*update*. Kemudian yang dilakukan hanyalah *train model* tersebut dengan data data baru lagi, tetapi *weight* pada *original model* ini harus tetap sama dan tidak berubah dan hanya *weight* pada layer baru saja yang ter-*update*.