Laporan ini dibuat untuk memenuhi persyaratan rekrutmen di Nexmedis Indonesia sebagai *Research and Development Staff*.

Disusun oleh Meitha Auliana Dwi Winarsih

I. Python Problem

Pada teknikal tes pertama ini, kandidat diminta untuk membuat sebuah sistem back-end sederhana menggunakan FastAPI dan Uvicorn. Hal yang harus diperhatikan selama pembuatan sistem ini adalah, minimal harus memuat root endpoint, sebuah endpoint untuk mengirim data JSON, dan sebuah endpoint untuk menerima data JSON.

Berdasarkan intruksi tersebut, saya berencana membuat halaman web sederhana yang memungkinkan user untuk menambahkan daftar pasien, menyimpan daftar tersebut, menampilkannya, dan menghapusnya.

Untuk mengakomodasi kebutuhan ini, saya mengimplementasikan REST API menggunakan Fast API untuk menangani data pasien. API ini memiliki 3 endpoint utama yaitu, `GET /patients/`, `POST /patients/`, dan `DELETE /patients/{index}`. Penerapan ini dapat dilihat pada kode yang ada di Tabel 1.

Endpoint `POST /patients/` digunakan untuk menghandle proses penambahan data baru ke database (`patients_db`). Data yang tersimpan di variabel `patients_db` disimpan dalam format JSON ini berupa pasangan *key-value* untuk nama, usia, jenis kelamin, dan diagnosis dari user (dokter). Setelah data pasien disimpan ke dalam database, fungsi save`_patients()` akan dipanggil. Hal ini dilakukan supaya data yang sudah tersimpan di sesi sebelumnya tetap akan ada meskipun server mati.

Di sisi lain endpoint `GET /patients/` berfungsi untuk mengambil keseluruhan data nama pasien yang telah tersimpan dalam database. Data terbaru yang disimpan di endpoint `POST /patients/` akan dimuat ulang setiap kali server di *refresh* ataupun dihidupkan kembali.

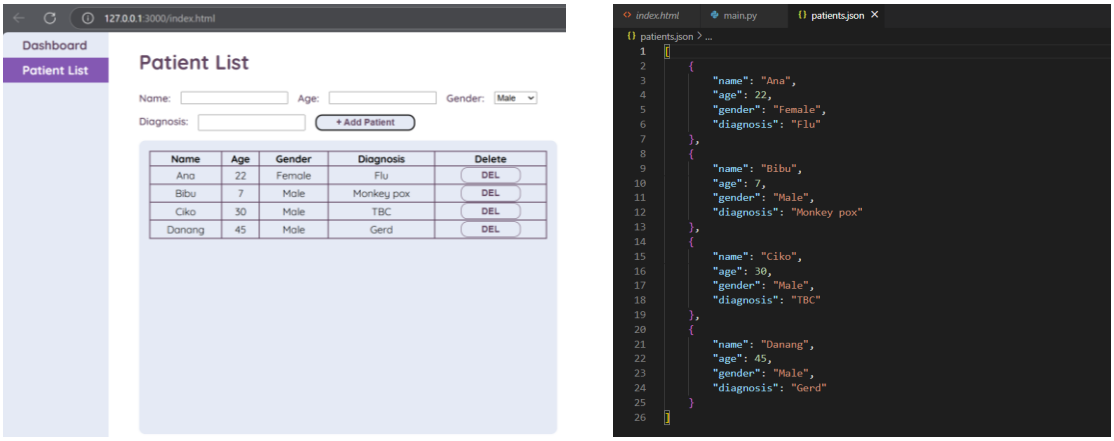
Tabel 1. Endpoint POST /patients/, GET /patients/, dan DELETE /patients/{index}

```
@app.post("/patients/")
async def add_patient(patient: Patient):
    patients_db.append(patient.model_dump())
    save_patients()
    return patient

@app.get("/patients/", response_model=List[Patient])
async def get_patients():
    return patients_db

@app.delete("/patients/{index}")
async def delete_patient(index: int):
    if index < 0 or index >= len(patients_db):
        raise HTTPException(status_code=404, detail="Patient not found")
    patients_db.pop(index)  # Hapus pasien dari list
    save_patients()
    return {"message": "Patient deleted"}
```

Untuk mendukung visualisasi sistem back-end, saya membuat halaman web sederhana berbasis HTML CSS. Halaman web tersebut dapat dilihat pada Gambar 1a. Di halaman ini, data JSON yang telah disimpan sebelumnya divisualisasikan dalam bentuk tabel. Sebagai perbandingan, Gambar 1b adalah data RAW berformat JSON yang disimpan di lokal direktori.



a. b.

Gambar 1. Front-end dan back-end sistem

Integrasi antara front-end dan back-end ini dihandle oleh JavaScript sederhana. Kode JavaScript mengelola pengiriman data pasien ke back-end melalui permintaan HTTP menggunakan metode POST. JavaScript akan mengambil data input dari tag HTML `<form>` dan dikumpulkan dalam sebuah objek `patientData`. Meskipun validasi tipe data sudah dilakukan di back-end menggunakan Pydantic BaseModel, skrip ini tetap menerapkan validasi tambahan di sisi front-end. Memastikan bahwa setiap input terisi dengan benar.

Setelahnya, sisi front-end akan mengirim permintaan HTTP POST ke API back-end dengan URL seperti yang tertera pada kode di Tabel 2. Dengan metode POST, front-end akan mengirimkan data baru ke back-end dalam format JSON.

Tabel 2.

```
document.getElementById("patient-form").addEventListener("submit", async
function(event) {
    event.preventDefault();

    let patientData = {
        name: document.getElementById("name").value,
        age: document.getElementById("age").value,
        gender: document.getElementById("gender").value,
        diagnosis: document.getElementById("diagnosis").value
    };

    if (!patientData.name || isNaN(patientData.age) || patientData.age <= 0
|| !patientData.gender || !patientData.diagnosis) {
        alert("Please fill in all fields correctly.");
        return;
    }
```

```
    // Kirim data ke backend
    let response = await fetch("http://127.0.0.1:8000/patients/", {
        method: "POST",
        headers: {
        "Content-Type": "application/json"
        },
        body: JSON.stringify(patientData)
     });
```

Untuk kemudahan modifikasi data di bagian front-end, ditambahkan tombol DEL yang akan menghapus data pasien dengan index tertentu dari server. Fungsi `deletePatient(index)` di front-end akan mengirim permintaan DELETE ke endpoint `DELETE /patients/{index}`, dengan index merepresentasikan posisi pasien dalam daftar. Setelah data berhasil dihapus dari server, database akan dimuat ulang.

II. Medical Imaging

Permasalahan kedua adalah *multiclass classification* untuk dataset X-Ray. Hal pertama yang saya lakukan sebelum menuju ke penentuan jenis arsitektur adalah dengan mempersiapkan dataset. Jika dilihat, tujuan dari permasalahan ini adalah klasifikasi 3 kelas. Sedangkan kelas PNUMONIA BACTERIAL dan PNEUMONIA VIRAL masih tergabung ke dalam satu folder. Jadi langkah pertama yang harus dilakukan adalah memisahkan kedua data ini.

Selain pengelompokan masing-masing kelas, dapat dilihat bahwa citra X-Ray yang ada pada dataset memiliki dimensi yang berbeda-beda. Hal ini nantinya akan diatasi dengan melakukan tahapan preprocessing berupa *resize image* ke dalam dimensi 256 × 256.

Tahapan preprocessing nantinya juga dilanjutkan dengan normalisasi nilai pixel, dari range 0 – 255 menjadi 0 – 1. Hal ini dilakukan untuk men-generalisasi data dan menghindari nilai ekstrim yang dapat berdampak ke model.

Proses augmentasi dengan ImageDataGenerator juga dilakukan pada dataset training untuk menambah variasi data. Proses ini memuat augmentasi *zoom, rotation, width_shift*, dan *height_shift*.

Untuk mencegah model training terlalu lama dengan nilai loss yang stagnan, diatur fungsi early_stopping untuk menghentikan proses training jika variabel `val_loss` tidak berubah selama 5 epoch. Variabel `val_loss` dianggap berubah jika memiliki selisih nilai minimal 10^{-7}. Selain itu juga diatur callback untuk memfaktorkan nilai LR dengan 0.2 jika variabel `val_loss` tidak berubah selama 2 epoch. Hal ini dilakukan untuk melakukan tuning *learning rate* secara otomatis selama proses training berlangsung.

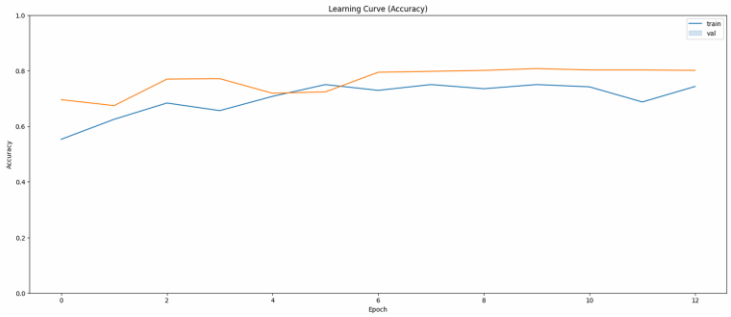
Model pertama sebagai uji coba menggunakan basic CNN dengna 4 layer konvolusi seperti tertera pada Gambar 2. Model pertama ini dilatih dengan batch_size = 32, learning rate awal = $3 \cdot 10^{-5}$, dan optimizer Adam.

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 256, 256, 3)	0
conv2d (Conv2D)	(None, 256, 256, 32)	896
max_pooling2d (MaxPooling2D)	(None, 128, 128, 32)	0
conv2d_1 (Conv2D)	(None, 128, 128, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 64)	0
conv2d_2 (Conv2D)	(None, 64, 64, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 128)	0
conv2d_3 (Conv2D)	(None, 32, 32, 256)	295,168
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 256)	0
flatten (Flatten)	(None, 65536)	0
dense (Dense)	(None, 256)	16,777,472
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 3)	771

Total params: 17,166,659 (65.49 MB)
Trainable params: 17,166,659 (65.49 MB)
Non-trainable params: 0 (0.00 B)

Gambar 2. Model pertama



Gambar 3. Akurasi training dan validasi model pertama

Model pertama ini memiliki rata-rata akurasi training diantara angka 55% – 75% dan akurasi validasi 67% – 80% seperti terlihat pada Gambar 3. Beberapa hal yang dapat menjadi penyebab hasil akurasi kurang maksimal adalah sebagai berikut :

- Model CNN terlalu sederhana untuk dataset ini. Dapat dilihat dari strukturnya yang hanya tersusun atas layer konvolusi dan layer max-pooling
- Akurasi training yang lebih rendah dari akurasi validasi dapat disebabkan oleh ketidakseimbangan jumlah data di TRAIN sedangkan jumlah data di VAL lebih seimbang
- Learning rate awal yang mungkin terlalu kecil dapat menyebabkan model berjalan dengan lambat untuk mencapai konvergensi

Model kedua dirancang dengan arsitektur yang hampir mirip dengan model pertama. Hanya saja, pada model kedua ini, jumlah filter di layer pertama di mulai dari 16 hingga 128. Menyebabkan model memiliki size yang lebih kecil. Pada awalnya, penggunaan jumlah filter yang kecil ini dilakukan dengan tujuan untuk mempercepat proses training. Namun demikian, arsitekturnya yang lebih sederhana dari model sebelumnya menyebabkan model ini memiliki akurasi validasi yang sedikit lebih rendah – sekitar 62% sampai 73% untuk data validasi dan angka 68% sampai 78% untuk data training. Namun demikian, dapat diihat bahwa akurasi training dari model kedua ini lebih stabil jika dibandingkan model

sebelumnya. Hal ini dapat terjadi karena learning rate yang digunakan di model ini adalah $3 \cdot 10^{-3}$. Sehingga, di awal epoch model bisa mencapai konvergensi sedikit lebih cepat.

Model ketiga dibuat dengan mengembangkan model sebelumnya. Pada model ini ditambahkan layer batch normalization, aktivasi, dan drop out untuk tiap blok. Arsitektur lebih detail dapat dilihat pada Gambar 4.

Model: "functional"

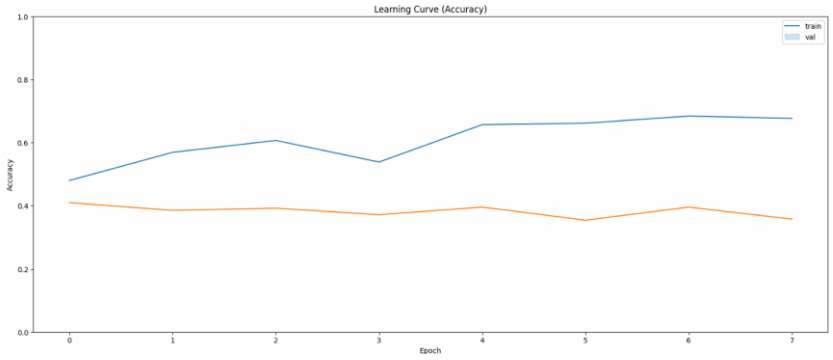
Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 256, 256, 3)	0
conv2d (Conv2D)	(None, 256, 256, 16)	448
batch_normalization (BatchNormalization)	(None, 256, 256, 16)	64
activation (Activation)	(None, 256, 256, 16)	0
max_pooling2d (MaxPooling2D)	(None, 128, 128, 16)	0
dropout (Dropout)	(None, 128, 128, 16)	0
conv2d_1 (Conv2D)	(None, 128, 128, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_2 (Conv2D)	(None, 64, 64, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 64, 64, 64)	256
activation_1 (Activation)	(None, 64, 64, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0
dropout_1 (Dropout)	(None, 32, 32, 64)	0
conv2d_3 (Conv2D)	(None, 32, 32, 128)	73,856
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 128)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 256)	8,388,864
dropout_2 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 3)	771

Total params: 8,487,395 (32.38 MB)
Trainable params: 8,487,235 (32.38 MB)
Non-trainable params: 160 (640.00 B)

Gambar 4. Model ketiga

Dari hasil training dengan bath-size 64 dan learning rate $3 \cdot 10^{-5}$, model ini memiliki nilai akurasi training dan validasi yang lebih kecil dibadingkan dua model sebelumnya. Hasil akurasi selama training dan validasi dapat dilihat pada Gambar 5. Hal ini mungkin dapat disebabkan oleh,

- Layer dropout yang diletakkan setelah masing-masing blok konvolusi menyebabkan kehilangan data terlalu awal.
- Ditambah lagi dengan learning rate yang cukup kecil, menyebabkan model semakin lama dalam mempelajari fitur
- Jumlah filter yang kecil di awal, yaitu 16



Gambar 5. Akurasi training dan validasi model ketiga

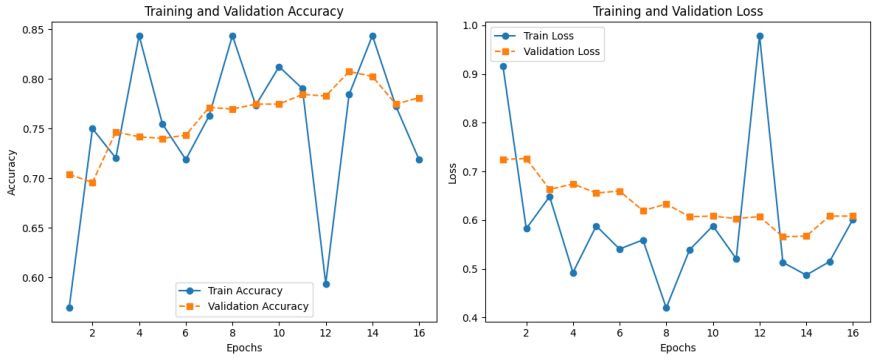
Sebagai uji coba terakhir karena model CNN yang di desain di atas tidak bekerja terlalu maksmial, digunakan backbone Xception dengan pre-trained weight dari imagenet untuk

model keempat. Arsitektur model lebih detail terlihat pada Gambar 6. Dari hasil training dengan bath-size 32 dan learning rate $3 \cdot 10^{-5}$, dapat dilihat bahwa model dengan pre-trained weights ini memiliki akurasi yang berfluktuasi pada rentang 70% - 85% seperti yang terlihat pada Gambar 7. Yang berarti memiliki akurasi yang sedikit lebih baik daripada custom model CNN sebelumnya.

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 256, 256, 3)	0
xception (Functional)	(None, 8, 8, 2048)	20,861,480
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dense (Dense)	(None, 128)	262,272
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 3)	387

Total params: 21,124,139 (80.58 MB)
Trainable params: 262,659 (1.00 MB)
Non-trainable params: 20,861,480 (79.58 MB)

Gambar 6. Arsitektur dengan backbone Xception



Gambar 7. Akurasi training dan validasi model dengan backbone Xception

III. Natural Language Processing

Permasalahan ketiga ini memanfaatkan *Natural Language Processing* (NLP) untuk memprediksi sentimen ulasan film ke dalam kategori sentimen positif atau negatif. Dataset IMDB ini memuat 50rb ulasan yang terbagi seimbang antara ulasan negatif dan positif.

Hal pertama yang saya lakukan adalah dengan membersihkan dataset. Berikut merupakan beberapa tahapan preprocessing yang dilakukan beserta tujuannya.

- Ubah semua teks ke dalam format *lowercase*. Hal ini dilakukan untuk menstandarisasi teks dengan kapitalisasi berbeda. Karena kita tidak ingin kata "good" dan "Good" diperlakukan sebagai kata yang berbeda oleh model
- Menghapus tag HTML yang tidak memiliki makna dalam teks, seperti <[^>]+>. Tag seperti ini tidak diperlukan karena tidak memiliki makna berarti terhadap data
- Menghapus tanda baca selain alfabet dan spasi
- Menghapus huruf-huruf yang berdiri sendiri seperti "a", "A", dll
- Menghapus bagian teks yang memiliki spacing lebih dari 1
- Menghapus stop words. Stop word adalah kata yang sering muncul dalam dokumen namun hanya memiliki makna semantik yang kecil. Kata *the, is, a, am*, dll

merupakan beberapa contoh dari stopwords. Dengan memanfaatkan fungsi stopwords dari library NLTK, model bisa lebih fokus pada kata-kata yang lebih bermakna. Selain itu, karena umumnya banyak ditemukan stopwords pada dokumen, penghapusan stopwords menyebabkan penurunan ukuran dataset yang signifikan. Kedua manfaat ini diharapkan dapat meningkatkan performa model. **Tetapi,** penghapusan stopwords juga tidak selamanya diperlukan. Bahkan kadang dapat mengubah makna dari teks. Oleh karena itu saya mencoba 2 model yang sama, satu model menggunakan stopwords default, dan model lain tidak memasukkan kata sentimen seperti *not, most, never* ke dalam stopwords.

Setelah melewati tahapan preprocessing, dataset yang sudah bersih akan diteruskan ke tahap *Tokenizer()*. Pada tahap ini, akan dibangun *dictionary* berdasarkan kata-kata yang ada dalam data `xtrain`. Setelah *dictionary* selesai dibuat, setiap kata yang ada di `xtrain` dan `xtest` akan diubah ke dalam bentuk sekuens berdasarkan data *dictionary*.

Untuk mempersiapkan data ke dalam neural network, data perlu diatur dimensinya. Dalam kasus ini, data yang lebih panjang dari 100 sekuens akan dipotong, sementara yang kurang dari 100 sekuens akan di padding.

Setelah datanya sudah siap, sekuens dari setiap kata akan diubah ke dalam bentuk data vektor menggunakan metode Word Embedding. Vektor ini menangkap makna semantik antar kata sehingga berguna dalam tugas seperti NLP. Database *GloVe 6B 100d* digunakan untuk mendapatkan vektor terlatih.

Jadi, setiap kata di *dictionary* yang juga muncul dalam database *GloVe* akan mengadopsi nilai vektor dari database tersebut. Data yang sudah disimpan dalam bentuk vector inilah yang akan digunakan sebagai input dari neural network.

Ada 3 model utama yang diujicoba, LSTM, CNN, dan gabungan CNN-LSTM. Gambar 8 merupakan arsitektur dari masing-masing model.

Model: "functional_8"

Layer (type)	Output Shape	Param #
input_layer_8 (InputLayer)	(None, 100)	0
embedding_8 (Embedding)	(None, 100, 100)	8,170,000
lstm_8 (LSTM)	(None, 128)	117,248
dense_8 (Dense)	(None, 1)	129

Total params: 8,287,377 (31.61 MB)
Trainable params: 117,377 (458.50 KB)
Non-trainable params: 8,170,000 (31.17 MB)

(a)

Model: "functional_12"

Layer (type)	Output Shape	Param #
input_layer_14 (InputLayer)	(None, 100)	0
embedding_16 (Embedding)	(None, 100, 100)	8,169,200
conv1d_7 (Conv1D)	(None, 100, 128)	38,528
global_max_pooling1d_1 (GlobalMaxPooling1D)	(None, 128)	0
dense_16 (Dense)	(None, 1)	129

Total params: 8,207,857 (31.31 MB)
Trainable params: 38,657 (151.00 KB)
Non-trainable params: 8,169,200 (31.16 MB)

(b)

Model: "functional_7"

Layer (type)	Output Shape	Param #
input_layer_7 (InputLayer)	(None, 100)	0
embedding_7 (Embedding)	(None, 100, 100)	8,169,200
conv1d_6 (Conv1D)	(None, 100, 128)	38,528
max_pooling1d_5 (MaxPooling1D)	(None, 50, 128)	0
lstm_7 (LSTM)	(None, 128)	131,584
dropout_6 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 1)	129

Total params: 8,339,441 (31.81 MB)
Trainable params: 170,241 (665.00 KB)
Non-trainable params: 8,169,200 (31.16 MB)

(c)

Gambar 8. Model (a) LSTM, (b) CNN, dan (c) CNN-LSTM

Ketiga model ini akan dilatih untuk menentukan model mana yang memiliki akurasi paling baik. Dari hasil ini, model terbaik akan diteruskan untuk masuk ke tahapan parameter tuning. Tabel 3 merupakan perbandingan akurasi dari masing-masing model.

Tabel 3. Perbandingan akurasi untuk 3 model

Parameter	LSTM	CNN	CNN-LSTM v1
Epoch	10	10	20
Batch Size	64	32	32
Learning Rate	0.001	0.0001	0.0001
Akurasi Test	0.8593	0.8504	0.8529

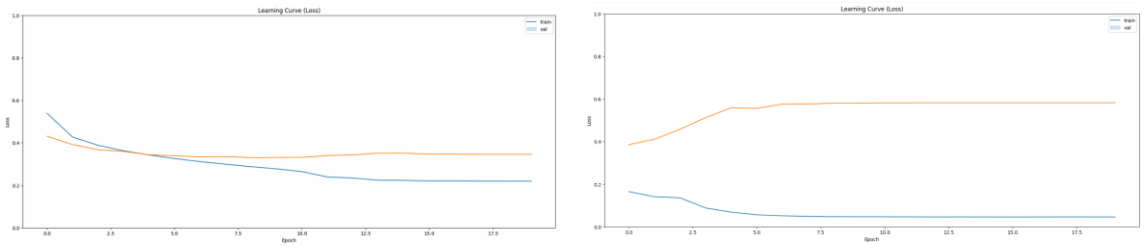
Karena ketiga model tidak memiliki perbedaan akurasi yang begitu signifikan, maka saya memutuskan untuk menggunakan model CNN-LSTM untuk dilanjutkan ke tahapan parameter tuning. Tabel 4 merupakan hasil parameter tuning dari model CNN-LSTM v1 di atas.

Tabel 4. Hasil tuning parameter

Parameter	CNN-LSTM v2	CNN-LSTM v3
Epoch	20	20
Batch Size	64	64
Learning Rate	0.001	0.0009
Akurasi Test	0.8661	0.8587

Dari hasil perbandingan akurasi untuk data testing di atas, dapat dilihat bahwa model CNN-LSTM v2 memiliki akurasi yang sedikit lebih baik dari versi lainnya. Selain itu, meskipun CNN-LSTM v3 memiliki akurasi yang sedikit lebih besar dari CNN-LSTM v1, tidak di sarankan untuk memilih v3 karena learning curve nya yang menunjukkan indikasi overfitting. Perbandingan learning curve untuk v1 dan v3 dapat dilihat pada Gambar 9.

Selain itu, model LSTM di latih sekali lagi untuk menguji hipotesa stopwords. Beberapa custom_stopwords yang dimasukkan ke dalam daftar adalah *'not', 'nor', 'no', 'never', 'none', 'nobody', 'nothing', 'neither', 'very', 'too', 'just', 'only', 'almost', 'quite', 'more', 'most', 'less', 'least', 'without', 'against'*.



(a) (b)

Gambar 9. Learning curve (a) CNN-LSTM v1 dan (b) CNN-LSTM v3

Dengan ini, kata-kata yang masuk ke dalam custom list di atas tidak akan dihapus dari teks. Dari hasil ujicoba, akurasi dari metode ini adalah 0.8592. Kurang lebih sama dengan model LSTM yang menggunakan default stopwords, dengan akurasi 0.8593. Artinya, dalam kasus ini penggunaan custom stopwords tidak meningkatkan akurasi model secara signifikan.