PENERAPAN ARSITEKTUR TEXTCNN UNTUK ANALISIS SENTIMEN ULASAN PENGGUNA APLIKASI SHOPEE DI GOOGLE PLAY STORE

TUGAS AKHIR

Diajukan sebagai syarat menyelesaikan jenjang strata Satu (S-1) di Program Studi Teknik Informatika, Fakultas Teknologi Industri, Institut Teknologi Sumatera

Oleh:
ELIKA EUGENIA RAMADHANIA
121140212

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA
LAMPUNG SELATAN
2025

RINGKASAN

Penerapan Arsitektur TextCNN Untuk Analisis Sentimen Ulasan Pengguna Aplikasi Shopee di Google Play Store

Elika Eugenia Ramadhania

Pertumbuhan *e-commerce* di Indonesia, khususnya aplikasi Shopee yang menduduki peringkat pertama sebagai toko *online* dengan kunjungan terbanyak, memerlukan analisis mendalam terhadap ulasan pengguna untuk memahami persepsi publik. Penelitian terdahulu menunjukkan ketidaksesuaian antara rating bintang dan isi ulasan, dimana sekitar 20% ulasan menunjukkan inkonsistensi antara teks dan rating numerik yang diberikan. Hal ini menunjukkan bahwa klasifikasi sentimen tidak dapat hanya mengandalkan rating bintang, tetapi perlu mempertimbangkan konteks isi ulasan. Rumusan masalah penelitian ini adalah bagaimana menerapkan dan mengukur kinerja arsitektur TextCNN dalam analisis sentimen terhadap ulasan pengguna aplikasi Shopee di Google Play Store berdasarkan evaluasi akurasi, presisi, *recall*, dan *f1-score* menggunakan *stratified k-fold cross validation*. Tujuan penelitian adalah mengimplementasikan arsitektur TextCNN untuk analisis sentimen ulasan Shopee dan mengevaluasi kinerjanya menggunakan *stratified k-fold cross validation*. Metodologi penelitian menggunakan dataset 5.000 ulasan aplikasi Shopee dari Google Play Store periode Juni sampai September 2024. *Preprocessing data* meliputi *case folding*, *cleansing*, *normalization*, *tokenizing*, *stopwords removal*, dan augmentasi teks. Empat model TextCNN dikembangkan yaitu TextCNN Ringan, Sedang, Berat, dan PAT-CNN. Evaluasi dilakukan menggunakan 5 *fold cross validation* dengan metrik akurasi, presisi, *recall*, dan *f1-score*. Hasil penelitian menunjukkan TextCNN Sedang

mencapai performa terbaik dengan akurasi 87.68%, presisi 89.85%, *recall* 93.88%, dan *f1-score* 91.81%. TextCNN Berat menghasilkan *f1-score* tertinggi 92.05%, sementara PAT-CNN menunjukkan presisi terbaik 92.21%. *Ablation study* mengidentifikasi *optimizer* Adam, *learning rate* 0.0001, dan *dropout* 0.4-0.5 sebagai konfigurasi optimal. Meskipun terdapat variasi performa antar model, perbedaannya relatif kecil dengan rentang akurasi 1.64%. Kesimpulan penelitian ini membuktikan bahwa arsitektur TextCNN dapat diterapkan secara efektif untuk analisis sentimen ulasan *e-commerce* dengan hasil yang memuaskan. *K-fold cross validation* terbukti efektif dalam mengukur konsistensi performa model. *Preprocessing data* komprehensif berkontribusi signifikan terhadap pencapaian performa optimal, sehingga TextCNN dapat digunakan sebagai solusi klasifikasi sentimen ulasan pengguna aplikasi dengan akurasi dan *reliability* yang baik.

ABSTRAK

Penerapan Arsitektur TextCNN Untuk Analisis Sentimen Ulasan Pengguna Aplikasi Shopee di Google Play Store

Elika Eugenia Ramadhania

Analisis sentimen ulasan pengguna aplikasi *e-commerce* menjadi penting untuk memahami persepsi publik, namun ketidaksesuaian antara rating numerik dan isi teks ulasan menunjukkan perlunya pendekatan berbasis pemrosesan bahasaalami. Penelitian ini bertujuan mengimplementasikan arsitektur TextCNN untuk analisis sentimen ulasan aplikasi Shopee di Google Play Store dengan evaluasi menggunakan *stratified k-fold cross validation*. Dataset berisi 5.000 ulasan periode Juni-September 2024 diproses melalui tahap *case folding*, *cleansing*, *normalization*, *tokenizing*, *stopwords removal*, dan augmentasi teks. Empat model TextCNN dikembangkan dengan variasi kompleksitas yaitu Ringan, Sedang, Berat, dan PAT-CNN. Evaluasi dilakukan menggunakan 5-*fold cross validation* dengan metrik akurasi, presisi, *recall*, dan *f1-score*. Hasil penelitian menunjukkan TextCNN Sedang mencapai performa terbaik dengan akurasi 87.68%, presisi 89.85%, *recall* 93.88%, dan *f1-score* 91.81%. TextCNN Berat menghasilkan *f1-score* tertinggi 92.05%, sementara PAT-CNN menunjukkan presisi terbaik 92.21%. Ablation study mengidentifikasi konfigurasi optimal menggunakan *optimizer* Adam, *learning rate* 0.0001, dan *dropout* 0.4-0.5. Penelitian ini membuktikan bahwa TextCNN dapat diterapkan secara efektif untuk klasifikasi sentimen ulasan *e-commerce* dengan konsistensi performa yang baik melalui *stratified k-fold cross validation*.

Kata Kunci: TextCNN, analisis sentimen, Shopee, *stratified k-fold cross validation*, *deep learning*

ABSTRACT

Application of TextCNN Architecture for Sentiment Analysis of Shopee App User Reviews on Google Play Store

Elika Eugenia Ramadhania

Sentiment analysis of e-commerce app user reviews is important for understanding public perception, but the discrepancy between numerical ratings and review text content indicates the need for a natural language processing based approach. This study aims to implement the TextCNN architecture for sentiment analysis of Shopee app reviews on the Google Play Store with evaluation using k-fold cross validation. The dataset contains 5,000 reviews from June to September 2024, which are processed through case folding, cleansing, normalization, tokenizing, stopwords removal, and text augmentation stages. Four TextCNN models were developed with varying levels of complexity are Light, Medium, Heavy, and PAT-CNN. Evaluation was conducted using 5-fold cross validation with accuracy, precision, recall, and f1-score metrics. The results show that Medium TextCNN achieved the best performance with an accuracy of 87.68%, precision of 89.85%, recall of 93.88%, and an f1-score of 91.81%. Heavy TextCNN produced the highest f1-score of 92.05%, while PAT-CNN showed the best precision of 92.21%. The ablation study identified the optimal configuration using the Adam optimizer, a learning rate of 0.0001, and dropout of 0.4-0.5. This study proves that TextCNN can be effectively applied to classify e-commerce review sentiment with good performance consistency through stratified k-fold cross validation.

Keywords: TextCNN, sentiment analysis, Shopee, k-fold cross validation, deep learning

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR RUMUS

DAFTAR KODE

BAB I
PENDAHULUAN

1.1 Latar Belakang Masalah

Inovasi dalam teknologi informasi dan komunikasi telah membawa perubahan signifikan dalam berbagai aspek kehidupan manusia, termasuk dalam bidang perdagangan dan layanan digital. Salah satu sektor utama yang mengalami perkembangan pesat adalah *e-commerce*. *E-commerce* memanfaatkan teknologi untuk memfasilitasi aktivitas jual beli melalui jaringan internet, sehingga memberikan kemudahan bagi masyarakat untuk memenuhi kebutuhan sehari- hari tanpa harus meninggalkan tempat tinggal mereka [1]. Di Indonesia, pertumbuhan sektor *e-commerce* telah menunjukkan tren yang sangat positif, didukung oleh penyebaran internet yang terus meningkat, serta adopsi perangkat mobile yang semakin luas di kalangan masyarakat.

Aplikasi *e-commerce* berperan besar dalam kehidupan sehari-hari masyarakat. Shopee merupakan salah satu contoh dari *e-commerce* yang ada di Indonesia. Dilansir dari situs Databoks Katadata, Shopee merupakan *e-commerce* yang menduduki peringkat pertama sebagai toko online dengan kunjungan terbanyak di Indonesia tahun 2023. Tercatat bahwa ada total 325 juta masyarakat Indonesia mengakses shopee pada periode kuartal satu – kuartal dua di tahun 2023 [2]. Kemudahan akses, beragam fitur yang disediakan, serta berbagai promosi yang ditawarkan menjadi daya tarik utama yang mendorong popularitas aplikasi ini [3]. Platform Google Play Store dipilih sebagai sumber data karena tingginya tingkat penggunaan di Indonesia, di mana sebagian besar pengguna *smartphone* mengunduh dan memberikan ulasan aplikasi melalui platform ini [4]. Namun, di balik keberhasilan tersebut terdapat faktor

lain yang juga penting untuk diperhatikan, yaitu ulasan atau *review* pengguna [5]. Ulasan dari pengguna memiliki peran penting dalam menciptakan persepsi publik terhadap kualitas layanan dan produk yang ditawarkan oleh aplikasi.

Ketidaksesuaian antara rating bintang dan isi ulasan pada platform e-commerce menunjukkan bahwa penilaian numerik tidak selalu merepresentasikan sentimen pengguna secara akurat [6]. Hal ini didukung oleh penelitian terdahulu terhadap 8.600 ulasan di Google Play Store, yang menemukan bahwa sekitar 20% ulasan menunjukkan ketidaksesuaian antara isi teks dan rating bintang yang diberikan. Bahkan, ulasan dengan rating 3 bintang yang secara numerik dianggap netral, sering kali mengandung muatan sentimen negatif, seperti keluhan terhadap pengiriman, layanan, atau kualitas produk [7]. Penelitian ini mengklasifikasikan rating 3 bintang sebagai sentimen negatif, sejalan dengan penelitian sebelumnya yang menyatakan bahwa ulasan netral sulit dikenali secara [8].Hasil ini menunjukkan bahwa klasifikasi sentimen tidak bisa hanya mengandalkan rating bintang, tetapi perlu mempertimbangkan konteks isi ulasan. Dalam konteks ini, analisis sentimen menjadi salah satu pendekatan yang relevan. Analisis sentimen memungkinkan identifikasi pola emosi atau opini yang terkandung dalam teks ulasan, sehingga memberikan wawasan mendalam tentang kepuasan atau keluhan pengguna [9]. Dengan menggunakan metode ini, perusahaan dapat mengklasifikasikan ulasan menjadi kategori tertentu, seperti positif dan negatif [10].

Ulasan pengguna seringkali bersifat subjektif, mengandung bahasa informal, atau bahkan bahasa campuran, yang dapat menyulitkan proses analisis secara otomatis [11]. Penelitian oleh Duong & Nguyen-Thi (2021) menunjukkan bahwa penggunaan dataset sentimen berukuran

2.380 hingga 5.000 sampel untuk eksperimen klasifikasi teks, dengan dukungan teknik preprocessing dan augmentation, dapat menghasilkan performa f1-score yang memadai secara konsisten. Hal ini mengindikasikan bahwa skala data sebesar 5.000 ulasan sudah cukup representatif untuk pelatihan dan evaluasi model TextCNN dalam penelitian ini [12]. Selain itu, ketersediaan dataset yang telah terstruktur dengan baik memungkinkan fokus analisis lebih diarahkan pada pemodelan dan evaluasi performa klasifikasi.

Beberapa penelitian sebelumnya yang terkait dengan penelitian saat ini, diantaranya penelitian yang berjudul 'Analisis Sentimen Pengguna pada Aplikasi Tokopedia Menggunakan Algoritma *Convolutional Neural Network*'. Dalam penelitian ini, akurasi model CNN pada data uji didapatkan sebesar 83%. Performa terbaik pada kelas negatif dengan *precision* 0.85, *recall* 0.92, dan *f1-score* 0.88 [13]. Salah satu tantangan utama dalam analisis sentimen terhadap data ulasan pengguna adalah keberagaman struktur kalimat, penggunaan bahasa informal, dan adanya ketidaksesuaian antara penilaian numerik (rating) dengan isi teks ulasan [6]. Hal ini menjadikan proses klasifikasi sentimen menjadi tidak sederhana dan memerlukan pendekatan yang mampu menangkap makna konteks secara menyeluruh. Untuk mengatasi tantangan tersebut, pendekatan berbasis *deep learning* semakin banyak digunakan karena kemampuannya dalam mengekstraksi fitur penting dari data teks secara otomatis. Berdasarkan penelitian yang dilakukan Nhan Cach Dang dan rekan-rekan, model *deep learning* telah terbukti mampu menangani kompleksitas pemrosesan bahasa alami yang tidak bisa ditangani secara efektif oleh metode konvensional (*machine learning* tradisional) [14].

Salah satu algoritma deep learning yang terbukti efektif dalam analisis teks adalah *Convolutional Neural Network*. Meskipun awalnya

dikembangkan untuk pengolahan citra, arsitektur CNN telah berhasil diadaptasi untuk pemrosesan bahasa alami (*Natural Language Processing*), salah satunya melalui pengembangan model TextCNN [15]. TextCNN secara khusus dirancang untuk menangani data berbasis teks dan telah banyak digunakan dalam tugas analisis sentimen karena kemampuannya dalam mengenali pola- pola lokal dalam kalimat [16]. TextCNN bekerja dengan memanfaatkan *embedding layer* untuk merepresentasikan kata-kata dalam bentuk vektor berdimensi tetap, sehingga hubungan makna dan keterkaitan antar kata dapat dipertahankan [17]. Selanjutnya, lapisan konvolusi satu dimensi (Conv1D) diterapkan untuk menangkap *n-gram features* yang muncul dalam teks, seperti sentimen positif atau negatif [18]. Proses ini kemudian dilanjutkan dengan *global max pooling* untuk menyaring fitur paling penting, sebelum akhirnya diproses oleh lapisan klasifikasi untuk menghasilkan prediksi sentimen [19].

Dalam upaya menjaga kestabilan performa model serta menghindari *overfitting*, evaluasi dilakukan melalui pendekatan *stratified k-fold cross validation* guna menguji kapabilitas model dalam mengenali pola pada data yang belum pernah dilatihkan sebelumnya [20]. Teknik ini membagi dataset menjadi k subset yang berbeda, di mana pada setiap iterasi, satu subset digunakan sebagai data uji dan sisanya digunakan sebagai data latih. Dengan demikian, *stratified k-fold cross validation* memungkinkan penilaian kinerja model secara menyeluruh terhadap variasi data yang ada. Selain itu, analisis performa model dilakukan secara lebih rinci menggunakan *confusion matrix* untuk melihat distribusi prediksi model terhadap kelas sentimen, serta menghitung metrik evaluasi seperti akurasi, presisi, *recall*, dan *f1-score* [21].

Berdasarkan latar belakang di atas dengan menggabungkan TextCNN, *stratified k-fold cross validation*, dan analisis evaluatif menggunakan *confusion matrix*, penelitian ini tidak hanya berfokus pada pengembangan model klasifikasi, tetapi juga pada pemahaman menyeluruh terhadap kekuatan dan keterbatasan model. Penelitian ini diharapkan mampu memberikan gambaran yang lebih komprehensif terhadap efektivitas TextCNN dalam mengklasifikasikan sentimen ulasan pengguna serta memberikan kontribusi dalam pengembangan sistem rekomendasi dan peningkatan kualitas layanan *e-commerce* berbasis umpan balik pengguna.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dijelaskan, rumusan masalah dalam penelitian ini adalah bagaimana menerapkan dan mengukur kinerja arsitektur TextCNN dalam analisis sentimen terhadap ulasan pengguna aplikasi Shopee di Google Play Store berdasarkan evaluasi akurasi, presisi, *recall*, dan *f1-score* menggunakan *stratified k-fold cross validation*?

1.3 Tujuan Penelitian

Berdasarkan rumusan masalah yang telah diuraikan, tujuan dari penelitian ini adalah untuk mengimplementasikan arsitektur TextCNN dalam melakukan analisis sentimen terhadap ulasan pengguna aplikasi Shopee yang diperoleh dari platform Google Play Store, serta mengevaluasi kinerja model tersebut dengan menggunakan teknik *stratified k-fold cross validation* berdasarkan metrik evaluasi berupa akurasi, presisi, *recall*, dan *f1-score*.

1.4 Manfaat Penelitian

Hasil penelitian ini diharapkan dapat memberikan manfaat sebagai berikut.

1. Meningkatkan *awareness* dan kepercayaan pengguna terhadap ulasan produk dengan menyediakan analisis sentimen yang objektif dan sistematis.
2. Memberikan wawasan mengenai klasifikasi opini publik, baik positif maupun negatif, yang dapat digunakan untuk mengevaluasi performa suatu aplikasi.
3. Menjadi referensi dan sumber pembelajaran dalam memahami penerapan algoritma *deep learning* pada analisis sentimen.

1.5 Batasan Masalah

Untuk memastikan penelitian ini berfokus, adapun beberapa batasan masalah yang diterapkan sebagai berikut.

1. Data yang digunakan dalam penelitian ini diambil dari ulasan aplikasi Shopee Indonesia di Google Play Store.
2. Ulasan yang digunakan diperoleh dari periode Juni sampai dengan September 2024 dengan jumlah sebanyak 5000 ulasan.
3. Penelitian ini hanya terbatas pada klasifikasi sentimen ke dalam dua kategori, yaitu positif dan negatif.
4. Penelitian ini hanya menggunakan *stratified k-fold cross validation* sebagai teknik validasi untuk mengevaluasi performa model.
5. Parameter evaluasi kinerja algoritma meliputi akurasi, presisi, *recall*, dan *f1-score*.
6. Penelitian ini tidak mencakup implementasi algoritma dalam lingkungan produksi atau integrasi dengan sistem *real-time*.

1.6 Sistematika Penulisan

Sistematika penulisan terdiri dari lima bab. Setiap bab terdapat sub-sub bab serta penjelasan yang disusun prosedural untuk memudahkan pembaca dalam memahami tulisan. Penjelasan masing-masing bab sebagai berikut.

1.6.1 Bab I Pendahuluan

Berisi uraian secara umum berdasarkan penelitian yang dilakukan. Pada bab ini terdapat latar belakang penelitian, rumusan masalah, tujuan penelitian, batasan masalah, manfaat penelitian dan sistematika penulisan.

1.6.2 Bab II Tinjauan Pustaka

Berisi penelitian terdahulu dengan menjelaskan masalah apa yang diangkat di penelitian terdahulu, metode yang digunakan, serta landasan teori yang digunakan dalam penyusunan penelitian ini.

1.6.3 Bab III Metode Penelitian

Berisi terkait langkah-langkah penyelesaian masalah dari tahap pengumpulan data, metode pengembangan yang digunakan, perhitungan dan pengolahan data serta evaluasi pengujian untuk acuan penelitian pada bab selanjutnya.

1.6.4 Bab IV Hasil dan Pembahasan

Berisi analisis hasil dari penelitian yang memuat rancangan inti pada bab sebelumnya, yaitu data yang didapatkan dari pengerjaan penelitian yang sudah dikerjakan.

1.6.5 Bab V Kesimpulan dan Saran

Berisi kesimpulan yang didapatkan dari hasil dan pembahasan terkait penelitian yang dilakukan, dapat juga berupa temuan yang didapatkan setelah melakukan penelitian atau analisis penelitian yang dilakukan. Selain itu, saran akan dimuat pada bab ini untuk pengembangan penelitian selanjutnya.

BAB II
TINJAUAN PUSTAKA

2.1 Tinjauan Pustaka

Penelitian ini menggunakan tinjauan pustaka dari beberapa Pustaka yang telah dilakukan. Tinjauan Pustaka ini dapat dilihat pada Tabel 2.1 Tinjauan Pustaka.

Penelitian yang dilakukan oleh Alip Maskhuri dan rekan penelitannya pada tahun 2025 dengan judul "Analisis Sentimen Pengguna pada Aplikasi Tokopedia Menggunakan Algoritma *Convolutional Neural Network*" [13]. Permasalahan dalam penelitian ini adalah ketidakseimbangan data kelas negatif mendominasi 56.41%, menyebabkan model kesulitan membedakan sentimen netral dan positif. Hasil penelitian menunjukkan performa terbaik pada kelas negatif dengan akurasi pada data uji sebesar 83%. Kelas paling sulit dibedakan netral dan positif akibat ketidakseimbangan data dengan rincian negatif (8.885), netral (3.860), dan positif (3.006).

Penelitian yang dilakukan oleh Yunendah Nur Fuadah dan rekan penelitiannya pada tahun 2024 yang berjudul "Optimasi *Convolutional Neural Network* dan *K-Fold Cross Validation* pada Sistem Klasifikasi Glaukoma" [20]. Permasalahan utama dalam penelitian ini adalah Glaukoma dapat menyebabkan kebutaan permanen jika tidak terdeteksi dan ditangani dengan tepat, sehingga diperlukan sistem klasifikasi otomatis yang akurat. Peneliti mengusulkan arsitektur CNN sederhana (5 layer konvolusi + 1 *fully connected*) dengan 5-*fold cross validation* dan optimasi *hyperparameter* untuk mengklasifikasikan citra fundus menjadi 5 kategori glaukoma (normal, early, moderate, deep, OHT), menghasilkan akurasi 100% yang setara dengan AlexNet namun lebih efisien. Parameter

optimal epoch 100, batch size 64, learning rate 0.001, dan optimizer Adam.

Penelitian yang dilakukan oleh Erwin Yudi Hidayat dan rekan penelitiannya pada tahun 2023 yang berjudul "Penerapan 1D-CNN untuk Analisis Sentimen Ulasan Produk Kosmetik Berdasarkan Female Daily Review" [22]. Penelitian ini berfokus pada analisis sentimen ulasan produk kosmetik, khususnya dari brand Emina, dengan tujuan mengidentifikasi sentimen (positif, netral, negatif) karena ulasannya sering kali tidak terstruktur dan kadang tidak sesuai dengan skor bintang yang diberikan. Hasil penelitian menunjukkan bahwa model terbaik dilakukan di eksperimen ke-12 menggunakan Conv1D(128,3), *Dropout*(0.5), Dense(64-32-16), GlobalMaxPooling1D, fungsi aktivasi ReLU, *learning rate* 0.008, *epoch* 60.

Penelitian yang dilakukan oleh Risca Naquitasia dan rekan penelitiannya pada tahun 2022 yang berjudul "Analisis Sentimen Berbasis Aspek pada Wisata Halal dengan Metode *Deep Learning*" [23]. Permasalahan dalam penelitian ini adalah kurangnya penelitian analisis sentimen berbasis aspek pada wisata halal di Asia, ditambah dengan keterbatasan dataset (1.947 ulasan), ketidakseimbangan data antar kelas, dan masih terjadinya kesalahan prediksi model yang disebabkan oleh *preprocessing* yang belum optimal. Hasil penelitian menunjukkan klasifikasi aspek CNN mencapai akurasi tertinggi 98.299% *(epoch 15)* dan klasifikasi sentimen CNN mencapai akurasi tertinggi 93.96% (*epoch* 14). Model CNN menunjukkan performa lebih baik dibandingkan CNN-BiLSTM pada kedua jenis klasifikasi.

Penelitian yang dilakukan oleh Izzatul Azizah serta rekan penelitiannya pada tahun 2023 yang berjudul "Analisis Sentimen Ulasan Pengguna Aplikasi Shopee di Google Play menggunakan Metode *Word*

Embedding dan *Long Short Term Memory* (LSTM)" [24]. Penelitian ini berfokus pada analisis sentimen ulasan pengguna aplikasi Shopee di Google Play Store menggunakan pendekatan *deep learning* dengan *word embedding* dan LSTM, untuk mengetahui persepsi pengguna terhadap layanan Shopee. Dibutuhkan metode yang mampu menangkap makna teks secara kontekstual dan akurat, untuk mengelompokkan sentimen secara otomatis. Hasil akurasi akhir sebesar 0.73 (73%) dan *f1-score* 0.82. Parameter terbaik *word embedding* dimensi 20, learning rate 0.0001, dan *hidden size* 8. Model menunjukkan performa baik dalam mendeteksi sentimen positif dan negatif.

Tabel 2.1 Tinjauan Pustaka

No	Penulis, Tahun, Judul	Metode	Data	Hasil
1	Analisis Sentimen Pengguna pada Aplikasi Tokopedia Menggunakan Algoritma *Convolutional Neural Network* **Alip Maskhuri, Tikaridha Hardiani [2025]**	*Convolutional Neural Network* (CNN) dengan pelabelan otomatis menggunakan Pre-trained IndoBERT	Google Play Store dan media sosil X (Twitter) dalam rentang waktu 2022 2024.	Akurasimodel CNN pada data uji: 83%. Performa terbaik pada kelas negatif (*precision*: 0.85, *recall*: 0.92, dan *f1-score*: 0.88).

No	Penulis, Tahun, Judul	Metode	Data	Hasil
2	Optimasi *Convolutional Neural Network* dan *K-Fold Cross Validation* pada Sistem Klasifikasi Glaukoma **Yunendah Nur Fuadah, Ibnu Dawan Ubaidullah, Nur Ibrahim, Fauzi Frahma Taliningsing, Nidaan Khofiya Sy, Muhammad Adnan Pramuditho [2022]**	CNN (5-layer konvolusi dan 1 *fully connected layer)* dan 5-*fold cross validation*	RIM-ONE (RI) dataset dengan 5000 citra fundus	Akurasi yang di dapatkan sebesar 100% serta nilai presisi, *recall, fi-score,* dan AUC *score* senilai 1.
3	Penerapan ID-CNN untuk Analisis Sentimen	ID-*Convolutional Neural Network* (ID-	Ulasan produk Emina di situs	Eksperimen dilakukan sebanyak 30 kali untuk

No	Penulis, Tahun, Judul	Metode	Data	Hasil
	Ulasan Produk Kosmetik Berdasar Female Daily Review **Erwin Yudi Hidayat, Devioletta Handayani [2023]**	CNN) dan *Word Embedding: Pre-Trained Word2Vec*	Female Dailt Review dengan jumlah data awal 11.19 ulasan, setelah *pre-processing* 11.084 ulasan	mencari konfigurasi terbaik dan didapatkan model terbaik pada eksperimen ke-12, serta akurasi terbaik pada *training accuracy* 84.64% dan *testing accuracy* 80.22%
4	Analisis Sentimen Berbasis Aspek pada WIsata Halal dengan Metode *Deep Learning* **Risca Naquistasia, Dhomas Hatta**	*Deep Learning* dengan arsitektur CNN *(Convolutional Neural Network)* dan CNN-BiLSTM	Ulasan berbahasa Inggris dari website TripAdvisor dengan jumlah 1.947 ulasan dari tempat	Klasifikasi aspek CNN mencapai 98.299% (epoch 15) dan klasifikasi sentimen CNN mencapai

No	Penulis, Tahun, Judul	Metode	Data	Hasil
	Fudholi, Lizda Iswari [2022]		wisata di negara-negara Asia	akurasi tertinggi 93.96% (epoch 14)
5	Analisis Sentimen Ulasan Pengguna Aplikasi Shopee di Google Play Menggunakan Metode *Word Embedding* dan *Long Short-Term Memory* (LSTM) **Izzatul Azizah, Imam Cholissodin, Novanto Yudhistira [2023]**	*Word Embedding* menggunakan Word2vec dan algoritma *Long Short-Term Memory* (LSTM)	Google Play Store (laman Shopee) dengna jumlah ulasan 1.000 ulasan (Februari 2023)	Hasil akurasi akhir sebesar 0.73 (73%) dan *fi-score;* 0.82. Model menunjukkan performa baik dalam mendeteksi sentimen positif dan negatif.

Berdasarkan Tabel 2.1, mayoritas penelitian sebelumnya telah menerapkan pendekatan CNN dan variannya dalam analisis sentimen

berbasis teks. Namun, sebagian besar masih memiliki keterbatasan, seperti tidak digunakannya arsitektur TextCNN secara spesifik, tidak adanya validasi silang, atau konteks data yang kurang relevan. Penelitian yang dilakukan oleh Alip Maskhuri (2025) menggunakan CNN pada ulasan Google Play Store, tetapi belum menerapkan TextCNN [13]. Sementara itu, penelitian yang dilakukan oleh Erwin Yudi Hidayat (2023) menggunakan 1D-CNN untuk ulasan kosmetik, namun tidak melibatkan validasi silang [22]. Penelitian yang dilakukan oleh Izzatul Azizah (2023) membahas Shopee, tetapi menggunakan LSTM dan dataset yang jauh lebih kecil [24]. Berdasarkan hal tersebut, belum ditemukan penelitian yang secara khusus menerapkan TextCNN pada ulasan pengguna Shopee di Google Play Store dengan pendekatan *k-fold cross validation* dan evaluasi menggunakan *confusion matrix*. Oleh karena itu, penelitian ini dilakukan untuk melengkapi keterbatasan studi sebelumnya dengan pendekatan yang lebih terfokus dan menyeluruh.

2.2 Dasar Teori

Pada penelitian ini penulis menggunakan beberapa dasar teori yang berfokus pada penerapan TextCNN untuk analisis sentimen.

2.2.1 Analisis Sentimen

Analisis sentimen adalah proses mengidentifikasi, mengekstraksi, dan mengklasifikasikan emosi, opini, atau sikap seseorang terhadap suatu subjek, seperti produk, layanan, atau peristiwa tertentu [25]. Teknik ini bertujuan untuk menentukan apakah sentimen yang diungkapkan bersifat positif, negatif, atau netral. Analisis sentimen sering digunakan dalam bidang seperti pemasaran, politik, dan analisis media sosial. Proses analisis sentimen biasanya melibatkan beberapa langkah berikut [26]:

1. Pengumpulan Data, dapat diambil dari berbagai sumber, seperti media sosial, ulasan produk, atau forum diskusi.

2. *Preprocessing* Data, data teks biasanya membutuhkan pembersihan, seperti menghilangkan simbol, angka, *stop words*, dan melakukan *stemming* atau *lemmatization*.
3. Ekstraksi Fitur, teknik seperti TF-IDF (*Term Frequency-Inverse Document Frequency*) atau *word embedding* digunakan untuk merepresentasikan teks sebagai vektor numerik.
4. Klasifikasi Sentimen, algoritma *machine learning* atau *deep learning* digunakan untuk memprediksi kategori sentimen.
5. Evaluasi Model, tahap untuk mengukur seberapa baik model dalam memprediksi sentimen pada data yang belum pernah dilihat sebelumnya.

2.2.2 Aplikasi Shopee

Shopee adalah platform *e-commerce* yang memungkinkan pengguna untuk membeli dan menjual produk secara online. Didirikan pada tahun 2015 oleh Sea Group, Shopee menjadi salah satu aplikasi belanja online terkemuka di Asia Tenggara, termasuk Indonesia [27]. Dengan fitur-fitur seperti Shopee Mall, Flash Sale, dan layanan ShopeePay, Shopee dirancang untuk menyediakan pengalaman belanja yang mudah, cepat, dan aman.

2.2.3 Google Play Store

Google Play Store adalah platform distribusi digital yang dikembangkan oleh Google untuk perangkat Android. Platform ini menyediakan berbagai aplikasi, game, buku, dan konten digital lainnya yang dapat diunduh atau dibeli oleh pengguna. Google Play Store memiliki beberapa fitur utama, seperti pencarian dan kategori yang memudahkan pengguna dalam menemukan aplikasi berdasarkan jenisnya, ulasan dan rating yang memberikan *feedback* dari pengguna, serta sistem keamanan melalui Google Play Protect yang memindai aplikasi untuk mencegah *malware* [28]. Selain itu, Google Play Store juga

menyediakan pembaruan otomatis untuk meningkatkan keamanan dan performa aplikasi serta berfungsi sebagai sarana bagi pengembang dalam mendistribusikan dan memonetisasi produk mereka.

2.2.4 *Data Labeling*

Data labeling berfungsi sebagai target output yang digunakan untuk membandingkan hasil prediksi model dengan nilai sebenarnya, sehingga model dapat belajar dari kesalahan yang terjadi selama pelatihan [29]. Dalam konteks analisis sentimen, data labeling bertujuan untuk mengkategorikan teks ulasan ke dalam kelas tertentu seperti positif dan negatif. Jika ulasan memiliki rating satu, dua, dan tiga bintang, maka ulasan tersebut akan dikategorikan sebagai ulasan negatif atau label 0. Sementara itu, jika ulasan memiliki rating empat atau lima bintang, maka ulasan tersebut akan dikategorikan sebagai ulasan positif atau label 1 [8]. Pendekatan ini sejalan dengan penelitian sebelumnya yang menunjukkan bahwa ulasan dengan rating tiga bintang sering kali mengandung sentimen negatif atau keluhan tersembunyi, sehingga lebih tepat diklasifikasikan ke dalam kategori negatif untuk meningkatkan akurasi klasifikasi dan representasi persepsi pengguna secara lebih realistis.

2.2.5 *Prepocessing Data*

Preprocessing data adalah tahap awal dalam pemrosesan data yang bertujuan untuk membersihkan, menyiapkan, dan mengubah data mentah menjadi format yang lebih terstruktur agar dapat digunakan dalam analisis atau model pembelajaran mesin. Proses ini penting untuk meningkatkan kualitas data, mengurangi *noise*, agar lebih mudah diolah oleh algoritma *deep learning* serta memastikan model bekerja optimal. Dalam *pre-processing* terbagi menjadi beberapa tahapan yaitu [30]:

1. *Case Folding*

Proses *case folding* adalah proses mengubah seluruh huruf dalam teks

menjadi huruf kecil. Tujuan adalah untuk menghindari perbedaan akibat penggunaan huruf kapital dan menyamakan format teks.

2. *Cleansing*

   Cleansing adalah proses membersihkan teks dari elemen-elemen yang tidak diperlukan seperti URL, karakter khusus, atau spasi.

3. *Normalization*

   Normalization adalah proses untuk memperbaiki kata-kata non formal menjadi bentuk yang lebih baku dan umum untuk digunakan, sehingga memudahkan dalam analisis data.

4. *Tokenizing*

   Tokenizing adalah proses memecah teks menjadi unit kata atau token yang lebih kecil agar lebih mudah dianalisis.

5. *Stopwords Removal*

   Stopwords removal adalah proses menghapus kata-kata umum yang sering muncul tetapi tidak memiliki makna signifikan dalam analisis.

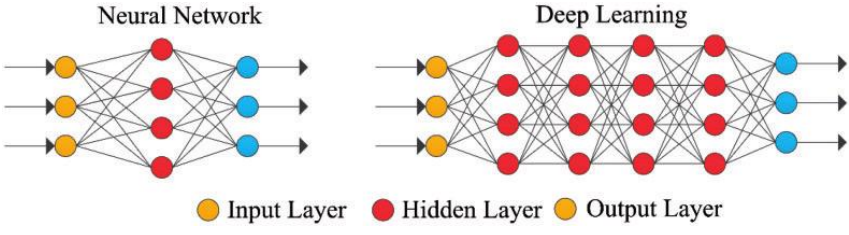
6. Augmentasi Teks

   Melakukan augmentasi pada teks yang sudah diproses seperti penggantian sinonim, *random typo*, *swap words*, dan *random delete*.

2.2.6 *Deep Learning*

Deep learning adalah cabang dari *Artificial Intelligence* (AI) atau kecerdasan buatan serta *machine learning* yang dikembangkan berdasarkan konsep *neural network* dengan beberapa lapisan (*multiple layers)*. Teknologi ini dirancang untuk meningkatkan ketepatan dalam berbagai tugas seperti deteksi objek, pengenalan suara, dan analisis teks [31]**.** Proses pelatihan melibatkan *forward propagation* untuk menghitung output yang *backward propogation* untuk memperbarui bobot berdasarkan kesalahan yang dihitung menggunakan fungsi *loss. Deep learning* sangat efektif dalam menangani data besar dan kompleks,

seperti gambar, teks, dan suara, berkat kemampuannya untuk secara otomatis mengekstrak fitur dari data tanpa memerlukan ekstraksi fitur manual [31]. Dengan kemajuan dalam komputasi, *deep learning* telah menjadi metode yang dominan dalam berbagai aplikas, termasuk visi komputer, pemrosesan bahasa alami, dan pengenalan suara, serta telah mendorong kemajuan signifikan dalam teknologi AI modern. Berikut arsitektur umum *deep learning* seperti pada Gambar 2.1 *Deep Learning Diagram* [32].



Gambar 2.1 *Deep Learning Diagram*

2.2.7 *Library* Python

Library Python merupakan kumpulan modul siap pakai yang menyediakan fungsi-fungsi khusus untuk mendukung proses pengolahan data, pembangunan model, serta evaluasi dalam penelitian ini. Pemanfaatan *library* untuk menghemat waktu pengembangan karena banyak algoritma, metode, dan utilitas telah tersedia secara optimal.

1. Library untuk Pemrosesan Data
    - Pandas untuk manipulasi dan analisis data terstruktur melalui DataFrame yang mendukung pembacaan file, *filtering*, dan transformasi kolom pada tahap *preprocessing* dataset ulasan [33].
    - NumPy adalah *library* komputasi numerik yang digunakan untuk operasi array multidimensi, perhitungan statistik, serta manipulasi data tensor [33].

2. *Library* untuk *Natural Language Processing*
    - NLTK (*Natural Language Toolkit*) adalah *platform* pemrosesan bahasa alami yang menyediakan korpus, *stopwords* bahasa Indonesia, serta fungsi tokenisasi untuk tahap *preprocessing* teks [34].
    - *Transformers* merupakan *library* dari Hugging Face yang digunakan melalui AutoTokenizer untuk memanfaatkan *tokenizer* IndoBERT dalam mengubah teks menjadi representasi numerik sesuai karakteristik bahasa Indonesia.
3. *Library* untuk *Deep Learning*
    - PyTorch adalah *framework deep learning* yang menjadi fondasi implementasi TextCNN dengan menyediakan modul-modul *neural network*, operasi tensor, dan mekanisme autograd.
    - TorchInfo untuk menampilkan ringkasan arsitektur PyTorch termasuk ukuran layer, jumlah parameter, dan kompleksitas komputasi.
    - Scikit-learn digunakan untuk implementasi StratifiedKFold serta evaluasi model melalui metrik akurasi, presisi, *recall*, *f1-score*, dan *confusion matrix* [33].
4. *Library* untuk Visualisasi dan Monitoring
    - Matplotlib adalah *library* visualisasi yang digunakan untuk membuat grafik metrik evaluasi dan *confusion matrix* [33].
    - Seaborn adalah *library* visualisasi statistik berbasis Matplotlib yang digunakan untuk membuat *heatmap confusion matrix* dengan tampilan yang lebih informatif [33].
    - Weights & Biases (wandb) adalah *platform experiment tracking* yang digunakan untuk *logging* metrik pelatihan, visualisasi performa, serta manajemen hasil model.

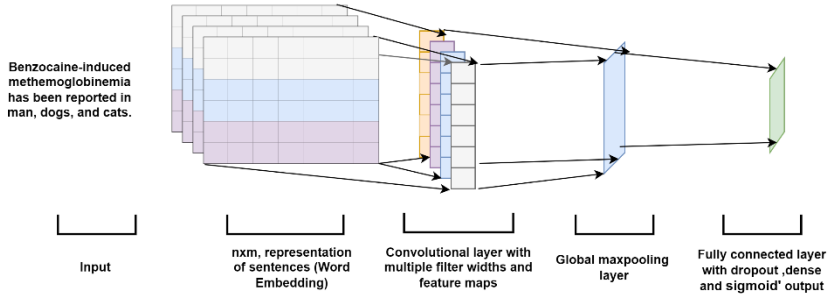
5. *Library* Utilitas
   - re (*Regular Expression*) adalah modul Python yang digunakan untuk pembersihan teks melalui pencarian dan penghapusan pola seperti URL, karakter khusus, dan spasi berlebih.
   - json adalah modul Python untuk menyimpan dan memuat konfigurasi fold, kamus normalisasi, serta hasil eksperimen dalam format terstruktur.
   - os adalah modul Python yang digunakan untuk manajemen direktori, pengecekan file, dan pembangunan *path* lintas *platform*.

2.2.8 *Text-based Convolutional Neural Network*

Text-based Convolutional Neural Network (TextCNN) merupakan salah satu arsitektur jaringan saraf tiruan yang dirancang khusus untuk menangani data berbasi teks. TextCNN merupakan adapatasi dari *Convolutional Neural Network* (CNN), yang awalnya dikembangkan untuk pengolahan citra, namun telah terbukti efektif dalam berbagai tugas pemrosesan bahasa alami *(Neural Language Processing)*, termauk analisis sentime [15]**.** Dalam arsitektur TextCNN, data masukan berupa teks terlebih dahulu diubah menjadi representasi vektor melalui *embedding layer* yang memungkinkan setiap kata direpresantikan dalam bentuk vektor berdimensi tetap. Selanjutnya, lapisan konvolusi satu dimensi (Conv1D) digunakan untuk mengekstraksi fitur-fitur lokal dari teks, seperti pola kata atau frasa yang memiliki makna sentimen tertentu (positif atau negatif) [16]**.** Proses konvolusi ini merupakan operasi aljabar linier yang dilakukan antara filter (kernel) dan vektor kata yang direpresantikan dalam bentuk matriks.

Lapisan konvolusi dalam TextCNN diikuti oleh fungsi aktivasi non-linear (biasanya ReLU) dan *global max pooling* untuk menyaring

fitur-fitur paling penting. Hasil dari beberapa filter konvolusi ini kemudian digabungkan (*concatenate),* sebelum proses oleh lapisan *fully connected* untuk menghasilkan output klasifikasi. Dengan struktur ini, TextCNN mampu mengenali pola n-gram yang bermakna dalam teks, dan menjadi salah satu metode yang efektif dalam klasifikasi sentimen ulasan pengguna [35]**.** Adapun arsitektur umum *Text-based Convolutional Network* yang sering digunakan seperti yang terlihat pada Gambar 2.2 Arsitektur Umum TextCNN [36]**.**



Gambar 2.2 Arsitektur Umum TextCNN

2.2.8.1 *Word Embedding*

Word embedding merupakan teknik representasi kata dalam bentuk vektor berdimensi tetap yang digunakan dalam pemrosesan bahasa alami. Memiliki tujuan untuk mengubah kata-kata dalam bentuk simbol (teks) menjadi representasi numerik (vektor) yang dapat diproses oleh model seperti TextCNN [37]**.** *Word embedding* memiliki sifat *dense* dan dapat merepresantikan hubungan makna dan struktur antar kata. Kata-kata yang memiliki makna serupa akan memiliki vektor yang saling berdekatan dalam ruang vektor. Dalam *deep learning embedding layer* dapat bersifat *trainable* yaitu bobot vektor *embedding* tidak berasal dari model *pre-trained*, melainkan diinisialisasi secara acak dan diperbarui selama proses pelatihan menggunakan algoritma *back-propogation* serta

cukup efektif dalam klasifikasi teks [38]**.** Pendekatan ini termasuk ke dalam skema *supervised learning,* karena pembaruan bobot *embedding* dilakukan berdasarkan *error* antara prediksi model dan label yang diketahui [39]**.**

Lapisan embedding menghasilkan representasi numerik dalam bentuk matriks dua dimensi yang berisi kumpulan vektor representasi dari setiap token dalam kalimat. Setiap baris pada matriks tersebut merepresentasikan satu token, sedangkan setiap kolom menunjukkan nilai fitur hasil embedding dengan panjang vektor yang tetap. Jumlah baris sesuai dengan banyaknya token dalam kalimat, dan jumlah kolom sesuai dengan dimensi embedding yang digunakan. Dengan demikian, setiap kalimat direpresentasikan sebagai sekumpulan vektor yang mencerminkan konteks dan hubungan semantik antar kata.

Untuk menyesuaikan format data agar dapat diproses oleh arsitektur Convolutional 1D, dilakukan operasi permute terhadap keluaran dari lapisan embedding. Operasi ini mengubah urutan dimensi tensor agar sesuai dengan format input yang diharapkan oleh Conv1D, yaitu dengan susunan dimensi [batch size, in channels, sequence length]. Pada tahap ini, setiap batch berisi satu atau lebih sampel teks, di mana setiap sampel memiliki panjang urutan token yang sama dan jumlah fitur embedding yang tetap. Hasil dari proses ini kemudian siap untuk diekstraksi oleh lapisan konvolusi untuk memperoleh pola-pola n-gram dalam teks.

2.2.8.2 *Convolutional Layer*

Convolutional layer merupakan inti dari arsitektur TextCNN yang berfungsi untuk mengekstraksi fitur-fitur lokal dari data input, seperti pola atau struktur spasial [15]. Proses konvolusi dilakukan

dengan menggeser filter (kernel) di sepanjang input dan menghitung hasil *dot product* antara filter dan *patch input* seperti pada (Persamaan 2.1.

$$y = \sum (W \odot X) + b \quad \text{(Persamaan 2.1)}$$

Keterangan:

y : Hasil dari operasi konvolusi (*output feature)*

W : Bobot kernel atau filter konvolusi

X : Input (input tensor sepanjang *sequence length)*

b : Bias

Setiap filter bertugas menangkap pola tertentu misalnya *edge,* bentuk, atau tekstur dalam data. Dalam konteks teks, *convolutional layer* mampu mengenali pola n-gram penting yang sering muncul dalam kalimat.

2.2.8.3 *Rectified Linear Unit (ReLU)*

Fungsi aktivasi *Rectified Linear Unit* (ReLU) digunakan setelah *convolutional layer* untuk menambahkan non-linearitas ke dalam jaringan. ReLU bekerja dengan mengubah semua nilai negatif menjadi nol dan mempertahankan nilai positif [40], dinyatakan dalam (Persamaan 2.2. Fungsi ini sangat populer karena sederhana, tidak mempengaruhi dimensi data dan mempercepat proses konvergensi dalam pelatihan model.

$$f(x) = \max(0, x) \quad \text{(Persamaan 2.2)}$$

Keterangan:

f(x) : Fungsi aktivasi ReLU

max(0, *x*) : Fungsi maksimum antara 0 dan *x*.

x : Input numerik dari setiap laye konvolusi ke fungsi aktivasi.

Fungsinya adalah memilih nilai maksimum antara 0 dan input *x,* dengan karakteristik sebagai berikut.

$$f(x) = \begin{cases} x, jika\ x > 0 \\ 0, jika\ x \leq 0 \end{cases}$$

2.2.8.4 *Global Max Pooling*

Global Max pooling adalah teknik *downsampling* yang digunakan untuk mengurangi dimensi spasial dari *feature map* yang dihasilkan oleh *convolutional layer*, sehingga memperkecil jumlah parameter dan mengurangi risiko *overfitting* [41]. Operasi ini bekerja dengan mengambil satu nilai maksimum dari seluruh *feature map* hasil konvolusi (1 dimensi) yang dilakukan dengan (Persamaan 2.3.

$$MaxPool(x) = max\ (x_1, x_2, \dots x_n) \qquad \text{(Persamaan 2.3)}$$

Keterangan:

MaxPool(x) : Operasi *pooling* maksimum pada vektor input *x*.

$x_1, x_2, \dots x_n$: Elemen-elemen dari vektor input.

max : Fungsi mengambil nilai maksimum dari seluruh elemen.

2.2.8.5 *Concatenate Features*

Concatenate features adalah proses menggabungkan output dari beberapa filter atau layer untuk membentuk representasi fitur yang lebih beragam. Tahap ini dilakukan setelah *max pooling*, di mana hasil dari berbagai ukuran kernel digabungkan menjadi satu vektor fitur sebagai input untuk lapisan berikutnya seperti pada (Persamaan 2.4).

$$f = [v^{(1)}; v^{(2)}; \dots; v^{(n)} \qquad \text{(Persamaan 2.4)}$$

Keterangan:

f : Vektor akhir hasil penggabungan semua fitur.

v : Vektor hasil ekstraksi fitur.

n : Jumlah dari input yang diproses.

Dimana nilai *f* merupakan anggota dari vektor dimensi $d_1 + d_2 + \cdots + d_n$ dengan *d* merupakan jumlah dari seluruh dimensi vektor.

2.2.8.6 *Dropout*

Dropout adalah teknik yang digunakan untuk mengurangi ketergantungan model terhadap fitur tertentu dengan cara menonaktifkan secara acak sebagian neuron selama pelatihan [42] seperti pada (Persamaan 2.5). Tahap ini dilakukan setelah *max pooling* dan *concatenation* untuk memastikan hasil ekstraksi fitur yang beragam tidak menyebabkan model terlalu fokus pada pola tertentu.

$$f_i' = \begin{cases} 0, & dengan\ probabilitas\ p \\ f_i, & dengan\ probabilitas\ 1-p \end{cases} \quad \text{(Persamaan 2.5)}$$

Keterangan:

f_i : Nilai aktivasi asli dari neuron ke-*i*.

f_i' : Nilai aktivasi neuro ke-*I* setelah diterapkan *dropout*.

P : Probabilitas neuron di-*drop* (dihilangkan/sementara dinonaktifkan).

$1-p$: Probabilitas neuron tetap aktif (dipertahankan nilainya).

2.2.8.7 *Fully Connected*

Fully connected layer (atau *dense layer*) adalah lapisan yang menghubungkan setiap neuron dari layer sebelumnya ke setiap neuron di layer ini. Lapisan ini berperan penting dalam tahap akhir proses klasifikasi, karena bertugas mengubah representasi fitur yang telah diekstraksi oleh layer sebelumnya menjadi nilai akhir yang digunakan untuk memprediksi kelas [42]. Tahap ini dimulai dengan mencari nilai logit menggunakan (Persamaan 2.6).

$$z = W \times f^{drop} + b \quad \text{(Persamaan 2.6)}$$

Keterangan:

z : Nilai logit atau output dari lapisan linear.

W : Bobot (*weight vector*) yang dipelajari selama proses pelatihan.

f^{drop} : Representasi hasil *dropout* dari fitur *f*.

b : Bias yaitu konstanta yang dipelajari selama pelatihan.

Nilai logit merupakan skor mentah yang belum dalam bentuk probabilitas. Untuk mengubah logit menjadi probabilitas yang dapat diuraikan, digunakan fungsi aktivasi *sigmoid* menggunakan (Persamaan 2.7).

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad \text{(Persamaan 2.7)}$$

Keterangan:

$\sigma(x)$: Fungsi aktivasi sigmoid menghasilkan output di antara 0 dan 1.

e^{-x} : Bilangan *Euler* (2,718), basis logaritma natural.

x : Input numerik (nilai logit).

Fungsi *sigmoid* akan memetakan nilai logit ke rentang (0,1), sehingga dapat melakukan prediksi kelas berdasarkan nilai probabilitas dengan membandingkan hasil *sigmoid* terhadap ambang batas 0.5, sebagai berikut.

$$\hat{y} = \begin{cases} 1, & jika\ \sigma(x) \geq 0.5 \\ 0, & jika\ \sigma(x) < 0.5 \end{cases} \qquad \text{(Persamaan 2.8)}$$

Keterangan:

$\hat{y}$: Hasil prediksi model (label kelas).

$\sigma(x)$: Hasil dari fungsi sigmoid yaitu probabilitas keluaran.

Jika $\sigma(x) \geq 0.5$, maka data diklasifikasikan ke kelas positif (label 1).

Jika $\sigma(x) < 0.5$, maka data diklasifikasikan ke kelas negatif (label 0).

Untuk mengukur selisih antara prediksi model dan label aktual, diterapkan fungsi *loss* yang sesuai untuk klasifikasi biner, yaitu *Binary Cross Entropy Loss* (BCELoss). Fungsi ini menghitung seberapa besar perbedaan antara nilai probabilitas hasil prediksi dengan label sebenarnya seperti (Persamaan 2.9).

$$L = -\left[y\ log(\hat{y})\ +\ (1-y)\ log(1-\hat{y})\right] \qquad \text{(Persamaan 2.9)}$$

Keterangan:

L : Nilai *loss* atau kesalahan prediksi.

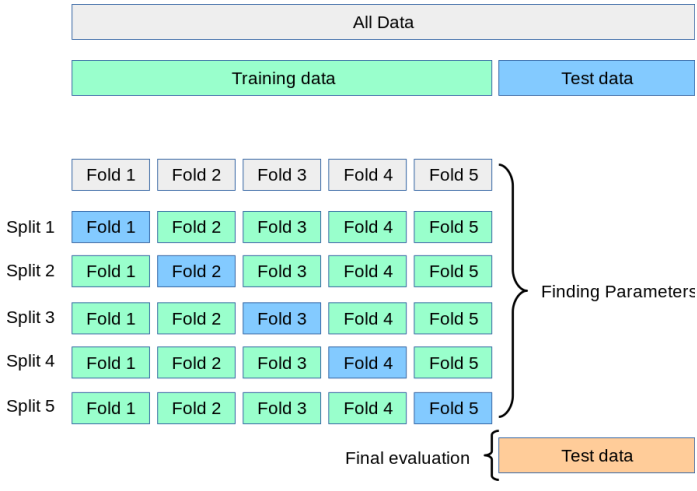
y : Label sebenarnya (*ground truth*), yaitu 0 atau 1.

$\hat{y}$: Prediksi probabilitas dari model, merupakan output fungsi sigmoid.

Nilai *loss* yang dihasilkan akan digunakan dalam proses *backpropagation*, yaitu metode propagasi kesalahan mundur dari output menuju parameter di setiap lapisan model untuk memperbarui bobot melalui algoritma optimisasi, seperti Adam. Proses ini memungkinkan model belajar memperbaiki kesalahan dan meningkatkan performa klasifikasi.

2.2.9 *Stratified K-Fold Cross Validation*

Stratified K-Fold Cross Validation merupakan pengembangan dari metode *K-Fold Cross Validation* yang dirancang khusus untuk menangani permasalahan distribusi kelas yang tidak seimbang pada dataset klasifikasi. Metode ini memastikan bahwa proporsi kelas pada setiap *fold* mencerminkan distribusi kelas keseluruhan dataset, sehingga setiap *fold* memiliki representasi kelas positif dan negatif yang seimbang [43]. Dengan demikian, evaluasi model menjadi lebih adil dan reliabel dibandingkan dengan *k-fold* standar, terutama ketika jumlah data antar kelas berbeda signifikan.



Gambar 2.3 Proses *Stratified K-Fold Cross Validation*

Proses *Stratified K-Fold* dilakukan dengan membagi dataset menjadi *k* lipatan (*fold*) yang relatif sama besar dengan mempertahankan proporsi label kelas pada setiap lipatan. Pada tiap iterasi, satu *fold* digunakan sebagai data uji, sementara *k* – 1 *fold* lainnya digunakan sebagai data latih. Proses ini diulang sebanyak *k* kali hingga seluruh *fold* memperoleh giliran sebagai data uji. Nilai evaluasi dari setiap iterasi kemudian dirata-ratakan untuk memperoleh estimasi performa model yang lebih stabil. Dalam konteks analisis sentimen, metode ini penting untuk menjaga keseimbangan distribusi label, misalnya antara ulasan positif dan negatif, sehingga hasil evaluasi lebih konsisten dan tidak bias. Umumnya nilai *k* yang digunakan adalah 5 atau 10, dengan metrik evaluasi berupa akurasi, presisi, *recall*, dan *f1-score* untuk menilai kemampuan generalisasi model.

2.2.10 *Confussion Matrix*

Confusion matrix adalah tabel evaluasi klasifikasi yang digunakan untuk mengevaluasi kinerja suatu sistem klasifikasi. Tabel ini berfungsi untuk mengevaluasi kinerja klasifikasi dengan membandingkan

label prediksi dan sebenarnya. *Confusion matrix* memiliki 4 istilah yang merepresentasikan hasil klasifikasi, yaitu:

1. *True Positive* (TP), jumlah contoh yang sebenarnya positif dan diklasifikasi benar sebagai positif.
2. *False Positive* (FP), jumlah contoh yang sebenarnya negatif tetapi diklasifikasi salah sebagai positif.
3. *True Negative* (TN), jumlah contoh yang sebenarnya negatif dan diklasifikasi benar sebagai negatif.
4. *False Negative* (FN), jumlah contoh yang sebenarnya positif tetapi diklasifikasi salah sebagai negatif.

Ke empat istilah ini merepresentasikan hasil klasifikasi benar (TP, TN) dan salah (FP, FN). Dengan *confusion matrix*, kita bisa menghitung matrik evaluasi seperti akurasi, presisi, *recall* menggunakan rumus yang melibatkan TP, FP, TN, FN seperti Tabel 2.2. Ini sangat berguna untuk menilai kinerja model klasifikasi.

Tabel 2.2 *Confussion Matrix* Biner

		Prediksi	
		Positif	**Negatif**
Aktual	**Positif**	*True Positive* (TP)	*False Negative* (FN)
	Negatif	*False Positive* (FP)	*True Negative* (TN)

Berdasarkan nilai TP, TN, FP, dan FN yang terdapat pada *confusion matrix*, dapat diperoleh beberapa parameter nilai akurasi, presisi, *recall*, dan *f1-score* yang digunakan untuk menghitung performa klasifikasi dari algoritma tersebut [21].

2.2.10.1 Akurasi

Akurasi adalah matriks utama yang biasa digunakan untuk menilai kinerja model klasifikasi. Akurasi menunjukkan proporsi sampel

yang diklasifikasi dengan benar dibandingkan total sampel [44]. Perhitungan nilai akurasi dapat dilihat pada (Persamaan 2.10)

$$Akurasi = \frac{TP + TN}{TP + FP + TN + FN} \quad \text{(Persamaan 2.10)}$$

2.2.10.2 Presisi

Presisi mengukur seberapa akurat prediksi positif dari sebuah model klasifikasi, yaitu seberapa banyak dari prediksi yang diklasifikasikan sebagai positif benar-benar positif [44]. Perhitungan nilai presisi dapat dilihat pada (Persamaan 2. 11).

$$Presisi = \frac{TP}{TP + FP} \quad \text{(Persamaan 2. 11)}$$

2.2.10.3 *Recall*

Recall juga dikenal sebagai *Sensitivity* atau *True Positive Rate*, adalah matrik evaluasi dalam klasifikasi yang mengukur kemampuan model untuk mendeteksi semua instance positif yang benar dalam data. Dengan kata lain, *recall* menunjukkan seberapa baik model dalam menemukan semua kasus positif yang ada [44]. Perhitungan nilai *recall* dapat dilihat pada (Persamaan 2.12).

$$Recall = \frac{TP}{TP + FN} \quad \text{(Persamaan 2.12)}$$

2.2.10.4 *F1-Score*

F1-Score adalah matriks evaluasi dalam klasifikasi yang menggabungkan *precision* dan *recall* dalam satu nilai untuk memberikan gambaran keseimbangan antara keduanya [44]. Nilai ini sangat berguna ketika terdapat ketidakseimbangan kelas dalam dataset, di mana hanya

mengandalkan *accuracy* saja bisa menyesatkan. Perhitungan nilai *f1-score* dapat dilihat pada (Persamaan 2.13).

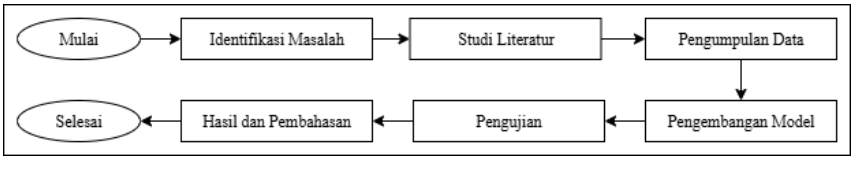
$$F1 - Score = 2 \times \frac{Presisi \times Recall}{Presisi + Recall} \qquad \text{(Persamaan 2.13)}$$

Nilai *F1-Score* berkisar antara 0 hingga 1, di mana 1 menunjukkan kinerja model yang sempurna, sedangkan 0 menandakan kinerja yang buruk. Matriks ini sangat penting dalam kasus seperti deteksi spam, analisis sentimen, dan deteksi anomali, di mana kesalahan klasifikasi dapat berdampak signifikan.

BAB III
METODE PENELITIAN

3.1 Alur Penelitian

Pada bagian ini akan dijelaskan tahapan-tahapan yang akan dilakukan dalam penelitian mulai dari awal hingga akhir penelitian. Tahapan-tahapan ini dibuat agar penelitian memiliki alur yang jelas dan terarah, serta memudahkan proses pengerjaan penelitian. Gambaran alur proses pengerjaan penelitian dapat dilihat pada Gambar 3.1 Alur Penelitian.



Gambar 3.1 Alur Penelitian

3.2 Penjabaran Langkah Penelitian

Pelaksanaan penelitian ini dilakukan sesuai dengan alur penelitian yang telah diilustrasikan dalam Gambar 3.1 Alur Penelitian. Setiap langkah dalam penelitian memiliki tujuan dan arti penting untuk memastikan proses penelitian berjalan dengan sistematis dan dapat dipahami secara menyeluruh, berikut adalah penjelasan yang lebih rinci mengenai masing-masing langkah tersebut.

3.2.1 Identifikasi Masalah

Penelitian dimulai dengan mengidentifikasi permasalahan terkait kebutuhan untuk memahami opini pengguna aplikasi Shopee berdasarkan ulasan pengguna di Google Play Store. Ulasan tersebut dapat berupa sentimen positif atau negatif, yang dapat mempengaruhi keputusan pengembangan aplikasi.

3.2.2 Studi Literatur

Pada tahap ini, dilakukan pencarian serta pengumpulan landasan teori dan metode apa yang dapat diterapkan untuk menyelesaikan permasalahan yang telah diidentifikasi sebelumnya. Studi literatur dilakukan dengan merujuk pada berbagai artikel dan jurnal yang terkait. Selain itu, penelitian ini juga mencakup analisis studi kasus serta perbandingan hasil dari penelitian terdahulu.

3.2.3 Pengumpulan Data

Pada penelitian ini, data dikumpulkan menggunakan data ulasan aplikasi Shopee yang berasal website Kaggle dimana data tersebut bersumber dari Google Play Store menggunakan teknik *web scraping*. Data yang dikumpulkan diambil dari periode Juni 2024 sampai dengan September 2024, yang mencakup pengguna, skor bintang, waktu, dan teks ulasan.

3.2.4 Pengembangan Model

Pada langkah ini, akan membahas model klasifikasi yang akan digunakan yaitu *Text-based of Convolutional Neural Network* yang dimulai dengan mempersiapkan data teks yang telah diproses dan dikonversi menjadi representasi numerik, seperti *word embedding*. Selanjutnya, arsitektur TextCNN dibangun dengan lapisan konvolusi untuk mengekstraksi fitur penting dari teks. Model ini kemudian dilatih menggunakan data latih dan dievaluasi menggunakan data uji untuk menilai performa klasifikasinya.

3.2.5 Pengujian

Pada tahapan ini model yang telah dibangun diuji menggunakan data uji untuk mengukur kinerjanya. Proses ini dilakukan dengan menghitung metrik evaluasi seperti akurasi, presisi, *recall*, dan *f1-score*

untuk menilai sejauh mana model mampu mengklasifikasikan sentimen dengan benar. Selain itu, metode *cross-validation* untuk memastikan model tidak *overfitting*.

3.2.6 Hasil dan Pembahasan

Pada tahap hasil dan pembahasan dilakukan pemaparan terhadap performa model berdasarkan hasil evaluasi yang telah diperoleh. Hasil klasifikasi sentimen ditampilkan dalam bentuk metrik seperti akurasi, presisi, *recall*, dan *f1-score* untuk menunjukkan efektivitas model dalam mengenali sentimen positif dan negatif.

3.3 Alat dan Bahan Tugas Akhir

Pada tahap hasil dan pembahasan dilakukan pemaparan terhadap performa model berdasarkan hasil evaluasi yang telah diperoleh. Hasil klasifikasi sentimen ditampilkan dalam bentuk metrik seperti akurasi, presisi, *recall*, dan *f1-score* untuk menunjukkan efektivitas model dalam mengenali sentimen positif dan negatif.

3.3.1 Alat

Adapun alat yang digunakan untuk mendukung proses penelitian adalah sebagai berikut.

1. *Hardware*

   Pada tugas akhir ini digunakan Laptop ASUS x415JP-A415JP dengan spesifikasi, sebagai berikut:

   a. Display: 14” FHD (1920 x 1080)
   b. Processor: Intel(R) Core(TM) i5-1035G1 1.19 GHz
   c. Memory: 8GB
   d. Storage: SSD NVMe 512GB
   e. Graphics: NVIDIA® GeForce MX330

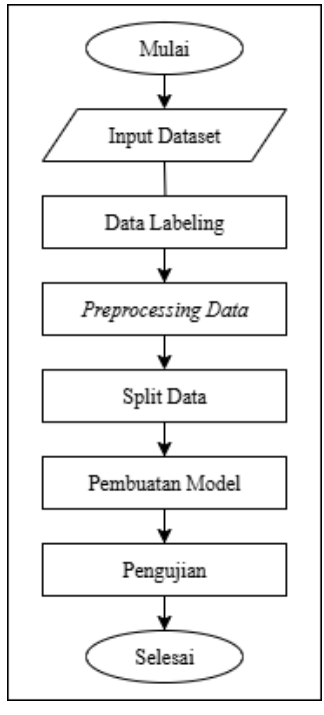
2. *Software*
    a. Visual Studio Code, digunakan untuk proses pengembangan dan eksperimen pemodelan.
    b. Python 3.12, bahasa pemrograman yang digunakan untuk pemrosesan data dan pengembangan model.
    c. Python Libraries digunakan untuk implementasi model dan visualisasi hasil.

3.3.2 Bahan

Bahan yang digunakan atau diperlukan untuk melakukan penelitian yaitu, dataset ulasan aplikasi Shopee dari Google Play Store yang berisi 5000 ulasan, dengan label pengguna, skor bintang, waktu, dan teks ulasan. Dataset ini diunggah oleh Nuri Cahyono di Kaggle dengan judul "Ulasan Shopee" yang diambil dari periode Juni 2024 sampai dengan September 2024.

3.4 Metode Pengembangan Model

Metode pengembangan yang digunakan yaitu model TextCNN (*Text-based of Convolutional Neural Network*). Model ini dirancang dengan tujuan untuk mengekstraksi fitur-fitur spasial lokal dari data sekuensial berupa teks, dengan cara menggabungkan representasi *word embedding* dan operasi konvolusi satu dimensi (1D *convolution*). Alur pengembangan model yang akan digunakan pada penelitian ini dapat dilihat pada Gambar 3.2 Alur Pengembangan Model.



Gambar 3.2 Alur Pengembangan Model

3.4.1 Input Dataset

Langkah awal dalam pengembangan sistem klasifikasi sentimen dimulai dengan input dataset teks ulasan pengguna aplikasi Shopee yang telah diperoleh untuk melatih model. Tahap ini penting karena kualitas dan kelengkapan data sangat memengaruhi hasil klasifikasi pada proses-proses berikutnya, seperti *preprocessing* dan pembagian data untuk pelatihan serta pengujian model.

3.4.2 Data Labeling

Tujuan dari melakukan analisis sentimen ialah melakukan pengelompokkan teks yang mengandung opini sentimen positif dan sentimen negatif. Maka pada tahap labeling data ini merupakan proses pemberian kategori pada dataset yang digunakan untuk mengelompokkan

data menggunakan kolom rating pada dataset yang digunakan. Sehingga, data labeling yang digunakan seperti pada Tabel 3.1 [8]:

Tabel 3.1 Ilustrasi Data Labeling Rating

	Rating	**Sentimen**
Label 0	1 2 3	Negatif
Label 1	4 5	Positif

Label ini dihasilkan secara otomatis dari kolom rating pada dataset, lalu dikonversi ke label biner. Pada Tabel 3.2 merupakan ilustrasi data labeling yang digunakan. Proses labeling ini bertujuan untuk menyederhanakan analisis klasifikasi sentimen menjadi dua kelas utama, sehingga lebih sesuai untuk model *binary classification* seperti TextCNN yang digunakan dalam penelitian.

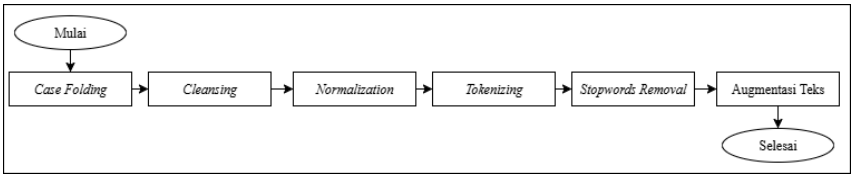
Tabel 3.2 Ilustrasi Data Labeling

Ulasan	**Rating**	**Label**
Mantap tempat perbelanjaan yang sangat praktis	5	Positif
Payah payah sudah 3x pengajuan tpi di tolak juga payah payah...sorry ya apk bakal gw apus	1	Negatif
Masa skrg gabisa pilih ekspedisi yg lain, mana lama banget sampenya	3	Negatif

3.4.3 *Prepocessing Data*

Pengolahan data awal atau *preprocessing* data merupakan hal yang penting untuk memastikan data yang telah didapatkan siap untuk dilakukan analisis lebih lanjut. Tahap ini bertujuan untuk memastikan bahwa data mentah yang diperoleh dalam hal ini berupa ulasan teks berada dalam

kondisi bersih, terstruktur, dan sesuai untuk dimasukkan ke dalam model pemrosesan bahasa alami. Tanpa proses preprocessing yang baik, model berisiko besar menangkap informasi yang salah, bias, atau tidak relevan. Proses ini mencakup beberapa langkah seperti Gambar 3.3 Alur *Preprocessing*.



Gambar 3.3 Alur *Preprocessing*

3.4.3.1 *Case Folding*

Case Folding merupakan salah satu tahap untuk menyamakan format penulisan dengan mengubah semua huruf menjadi huruf kecil (*lowercase*). Misalnya, kata ”Shopee”, ”shopee”, dan ”SHOPEE” akan diubah menjadi ”shopee”, sehingga dapat meningkatkan konsistensi dalam analisis teks.

Tabel 3.3 Proses *Case Folding*

Sebelum	Sesudah
Mantap tempat perbelanjaan yang sangat praktis	mantap tempat perbelanjaan yang sangat praktis
Payah payah sudah 3x pengajuan tpi di tolak juga payah payah...sorry ya apk bakal gw apus	Payah payah sudah 3x pengajuan tpi ditolak juga payah payah sorry ya apk bakal gw apus
Masa skrg gabisa pilih ekspedisi yg lain, mana lama banget sampenya	masa skrg gabisa pilih ekspedisi yg lain, mana lama banget sampenya

3.4.3.2 *Cleansing*

Pada tahap ini dilakukan penghapusan karakter yang tidak relevan seperti tanda baca, angka, simbol khusus, atau tautan. Contoh:

”Belanja di Shopee!!! #Diskon50%” menjadi ”Belanja di Shopee”. Memiliki tujuan untuk menghilangkan noise dalam teks agar data lebih bersih dan relevan untuk analisis.

Tabel 3.4 Proses *Cleansing*

Sebelum	Sesudah
mantap tempat perbelanjaan yang sangat praktis	mantap tempat perbelanjaan yang sangat praktis
payah payah sudah 3x pengajuan tpi di tolak juga payah payah sorry ya apk bakal gw apus	payah payah sudah pengajuan tpi di tolak juga payah payah sorry ya apk bakal gw apus
masa skrg gabisa pilih ekspedisi yg lain, mana lama banget sampenya	masa skrg gabisa pilih ekspedisi yg lain mana lama banget sampenya

3.4.3.3 *Normalization*

Tahap ini menstandarkan kata-kata tidak baku atau slang ke dalam bentuk formal. Dengan tujuan untuk mengurangi variasi dalam teks yang disebabkan oleh perbedaan ejaan atau penulisan tidak baku. Seperti ”gak” menjadi ”tidak”, ”bgt” menjadi ”banget”.

Tabel 3.5 Proses *Normalization*

Sebelum	Sesudah
mantap tempat perbelanjaan yang sangat praktis	mantap tempat perbelanjaan yang sangat praktis
payah payah sudah pengajuan tpi di tolak juga payah payah sorry ya apk bakal gw apus	payah payah sudah pengajuan tapi di tolak juga payah payah sorry
masa skrg gabisa pilih ekspedisi yg lain mana lama banget sampenya	masa sekarang tidak bisa pilih ekspedisi yang lain mana lama banget sampai nya

3.4.3.4 *Tokenizing*

Dalam langkah ini, teks ulasan dipecah menjadi potongan-potongan kata (token) yang lebih kecil. Contohnya mengubah "Shopee aplikasi belanja terbaik" menjadi ["Shopee", "aplikasi", "belanja", "terbaik"]. Tokenisasi memudahkan analisis lanjut, karena memungkinkan pemrosesan setiap kata secara individual, yang penting dalam menganalisis sentimen.

Tabel 3.6 Proses *Tokenizing*

Sebelum	Sesudah
mantap tempat perbelanjaan yang sangat praktis	['mantap','tempat', 'perbelanjaan', 'yang', 'sangat', 'praktis']
payah payah sudah pengajuan tapi di tolak juga payah payah sorry ya aplikasi bakal saya hapus	'pengajuan', 'tapi', 'di', 'tolak', 'juga', 'payah', 'payah', 'sorry', 'ya', 'aplikasi', 'bakal', 'saya', 'hapus']
masa sekarang tidak bisa pilih ekspedisi yang lain mana lama banget sampai nya	['masa', 'sekarang', 'tidak', 'bisa', 'pilih', 'ekspedisi', 'yang', 'lain', 'mana', 'lama', 'banget', 'sampai', 'nya']

3.4.3.5 *Stopwords Removal*

Stopwords adalah kata-kata umum yang tidak memberikan kontribusi signifikan dalam analisis sentimen, seperti "dan", "yang", dan "adalah". Dengan menghapus kata-kata ini, fokus analisis dapat diarahkan pada kata-kata yang lebih bermakna.

Tabel 3.7 Proses *Stopwords Removal*

Sebelum	Sesudah
['mantap','tempat', 'perbelanjaan', 'yang', 'sangat', 'praktis']	['mantap','tempat', 'perbelanjaan', 'praktis']

Sebelum	Sesudah
’pengajuan’, ’tapi’, ’di’, ’tolak’, ’juga’, ’payah’, ’payah’, ’sorry’, ’ya’, ’aplikasi’, ’bakal’, ’saya’, ’hapus’]	[’payah’, ’payah’, ’pengajuan’, ’tolak’, ’payah’, ’payah’, ’sorry’, ’aplikasi’, ’bakal’, ’hapus’]
[’masa’, ’sekarang’, ’tidak’, ’bisa’, ’pilih’, ’ekspedisi’, ’yang’, ’lain’, ’mana’, ’lama’, ’banget’, ’sampai’, ’nya’]	[’masa’, ’sekarang’, ’bisa’, ’pilih’, ’ekspedisi’, ’lain’, ’lama’, ’banget’, ’sampai’]

3.4.3.6 Augmentasi Teks

Augmentasi teks adalah proses memperkaya data dengan memodifikasi teks asli menggunakan berbagai teknik transformasi, tanpa mengubah makna secara signifikan. Tujuannya adalah meningkatkan variasi data pelatihan agar model lebih robust dan mampu mengenali pola secara lebih baik. Teknik augmentasi dapat meliputi:

1. Replace Phrases, mengganti frasa tertentu dengan bentuk lain yang memiliki makna serupa. Misalnya, “pengiriman sangat cepat” menjadi “pengantaran begitu cepat”.
2. Synonyms, mengganti kata dengan sinonimnya seperti “murah” menjadi “terjangkau”.
3. Random Typo, menambahkan kesalahan ketik secara acak seperti “bagus” menjadi “bagis”.
4. Random Swap, menukar posisi dua kata secara acak seperti “barang bagus” menjadi “bagus barang”.
5. Random Delete, menghapus kata secara acak seperti “produk ini sangat bagus” menjadi “produk sangat bagus”.

Dengan teknik augmentasi yang digunakan satu kalimat dapat menghasilkan beberapa versi berbeda, sehingga memperkaya korpus

data untuk pelatihan model analisis sentimen. Augmentasi dilakukan secara seri, yaitu setiap jenis transformasi diterapkan satu per satu secara berurutan pada teks yang sama. Pendekatan augmentasi secara seri ini dinilai cukup efektif karena mampu memperkaya variasi data sekaligus memperkuat kemampuan model dalam melakukan generalisasi terhadap input yang bervariasi. Sementara itu, pada *data validation* seluruh probabilitas augmentasi disetel ke nol sehingga *data validation* tetap dalam bentuk aslinya tanpa modifikasi. Pendekatan ini bertujuan agar model belajar dari variasi data yang lebih luas selama pelatihan, namun tetap dievaluasi secara objektif pada data yang belum dimodifikasi. Dengan demikian, augmentasi hanya diterapkan pada *data training* sebagai strategi untuk meningkatkan ketahanan model terhadap variasi struktur dan gaya bahasa dalam teks ulasan pengguna.

3.4.4 Split Data

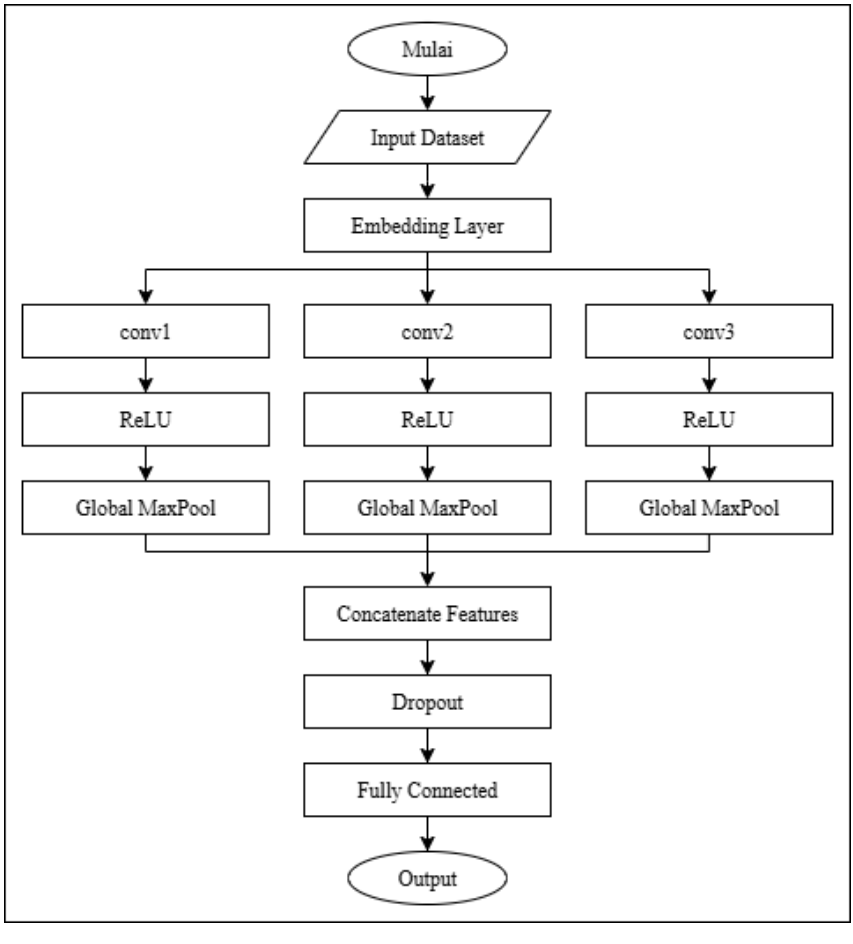
Pada penelitian ini, data dibagi menggunakan 5-fold cross validation dimana data dibagi menjadi 5 bagian. Setiap eksperimen atau training, 4 bagian 80% digunakan untuk training dan 1 bagian 20% untuk validasi, tetapi bagian validasi akan bergantian sehingga setiap data pernah menjadi data validasi. Sehingga rasio train:validasi pada setiap fold 80:20, tetapi pembagian ini dilakukan secara bergantian sebanyak 5 kali.

3.4.5 Pemodelan

Pemodelan dalam penelitian ini menggunakan arsitektur TextCNN untuk mengekstraksi fitur spasial dari data teks ulasan pengguna. Pada penelitian terdahulu, model ConvNet berbasis karakter berhasil dilatih dari awal (*training from scratch*) dan menunjukkan performa kompetitif pada berbagai dataset berskala besar [45]. Sehingga pada penelitian ini, model TextCNN dilatih dengan pendekatan *training from scratch* yaitu tanpa

menggunakan bobot pra-latih (*pretrained weights*). Seluruh parameter model, termasuk bobot pada *embedding layer*, diinisialisasi secara acak dan dioptimasi selama proses pelatihan menggunakan algoritma *backpropagation*. Pendekatan ini memungkinkan model mempelajari representasi fitur yang relevan secara langsung dari data yang digunakan.

Model terdiri dari *word embedding*, beberapa *convolutional layer* dengan ukuran kernel berbeda, *global max pooling*, dan *fully connected layer*. Ilustrasi model yang digunakan dapat dilihat pada Gambar 3.4.



Gambar 3.4 Ilustrasi Arsitektur Diagram TextCNN

Berikut merupakan penjelasan mengenai langkah-langkah pembuatan model berdasarkan Gambar 3.4 Ilustrasi Arsitektur Diagram TextCNN.

1. Input Data

   Model menerima input berupa tensor berdimensi [batch_size, sequence_length] yang merepresentasikan urutan kata dari sebuah kalimat. Setiap kata telah dikonversi menjadi indeks sesuai dengan kosakata yang dibangun dari data pelatihan.

2. *Word Embedding*

   Pada penelitian ini, digunakan *trainable embedding layer* dari PyTorch, layer ini mengubah indeks kata menjadi vektor berdimensi tetap (in_channels) yang menghasilkan output tensor berdimensi tiga dengan format [batch_size, sequence_length, in_channels]. Bobot *embedding* pada layer ini diinisialisasi secara acak dan diperbarui selama proses pelatihan menggunakan algoritma *back-propagation*, sehingga termasuk ke dalam pendekatan *supervised learning*. Untuk menyesuaikan format input dengan kebutuhan arsitektur *Convolutional* 1D, tensor hasil *embedding* kemudian dilakukan operasi permute, yaitu penukaran urutan dimensi, sehingga menjadi [batch_size, in_channels, sequence_length] sebelum diproses oleh lapisan konvolusi satu dimensi.

3. *Convolutional Layer*

   Tiga buah layer konvolusi 1D dengan ukuran kernel yang berbeda (3, 4, dan 5) digunakan secara paralel. Masing-masing layer konvolusi menghasilkan n fitur, sehingga total output dari tahap ini adalah nx3 fitur. Ketiga layer ini bertujuan untuk menangkap pola n-gram dari teks seperti 3-gram, 4-gram, dan 5-gram.

4. *Activation Function* (ReLU)

   Hasil dari setiap konvolusi diteruskan ke fungsi aktivasi ReLU (*Rectified Linear Unit*) untuk menambahkan non-linearitas ke dalam model.

5. *Global Max Pooling*

   Output dari setiap layer ReLU kemudian diproses menggunakan *Global Max Pooling* untuk mengekstrak fitur paling dominan dari setiap fitur map. Proses ini mereduksi dimensi sekuens dan hanya mempertahankan nilai maksimum dari setiap fitur.

6. *Concatenate Features*

   Ketiga hasil pooling dari setiap filter ukuran kernel kemudian digabungkan (*concatenate*) menjadi satu tensor berdimensi (batch_size, in_channels).

7. *Dropout*

   Untuk mencegah *overfitting*, digunakan *dropout* pada layer ini. Proses ini akan secara acak menonaktifkan sebagian unit selama pelatihan.

8. *Fully Connected Layer*

   Hasil dari *dropout* kemudian diteruskan ke layer linear (*fully connected*) dengan ukuran input n dan output sejumlah kelas (dalam hal ini 2 kelas: positif dan negatif). Layer ini bertugas melakukan klasifikasi berdasarkan fitur yang telah diekstraksi.

3.4.5.1 Fungsi *Loss*

Hasil prediksi dibandingkan dengan label target menggunakan fungsi *Binary Cross Entropy Loss* untuk menghitung tingkat kesalahan prediksi model. Proses pelatihan dilakukan dengan mekanisme *forward pass* di mana data input dikirim ke dalam jaringan untuk menghasilkan prediksi. Selanjutnya, prediksi tersebut dibandingkan dengan label aktual untuk menghitung nilai *loss*. Nilai *loss* ini merepresentasikan seberapa jauh

prediksi model dari nilai sebenarnya [46]. Proses dilanjutkan dengan *backpropagation*, yaitu perhitungan gradien dari nilai loss terhadap semua parameter model [47]. Gradien ini menunjukkan bagaimana setiap parameter berkontribusi terhadap kesalahan model dan akan digunakan oleh algoritma optimisasi untuk memperbarui bobot model. Dengan menggunakan skema tersebut, model diharapkan mampu meminimalkan error dan meningkatkan kemampuan dalam mengklasifikasikan sentimen ulasan pengguna secara akurat.

3.4.5.2 Hyperparameter

Pada penelitian menggunakan 3 model (ringan, sedang, berat) dimana setiap model menggunakan hyperparameter yang sama yaitu menggunakan *batch size* = 16, *learning rate* = 0.0004, dan nilai *dropout* = 0.2 yang sama serta menggunakan *optimizer* Adam. Optimizer Adam mengombinasikan keunggulan dari metode Momentum dan RMSProp dengan menghitung estimasi rata-rata pertama (mean) dan kedua (varians) dari gradien secara adaptif untuk setiap parameter [48]. Perbedaan model terletak pada variasi ukuran *embedding*, jumlah filter pada lapisan Conv1D, dan dimensi pada lapisan linear, yang memengaruhi tingkat kompleksitas dan kebutuhan komputasi masing-masing model. Variasi ukuran *embedding*, jumlah filter Conv1D, dan ukuran layer linear dapat dilihat pada Tabel 3.8.

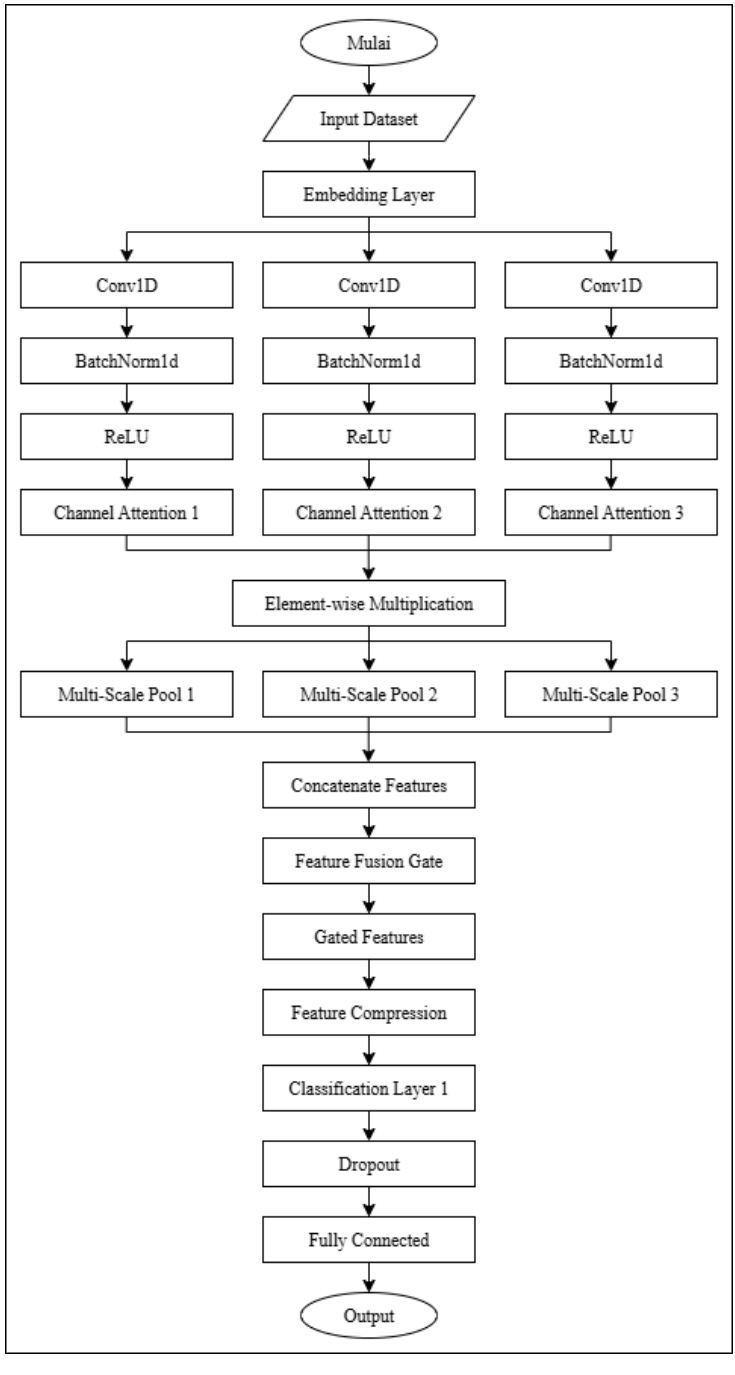
Tabel 3.8 Tabel Variasi Hyperparameter

Model	Embedding	Jumlah Filter	Ukuran Linear
TextCNN Ringan	100	50	150
TextCNN Sedang	300	100	300
TextCNN Berat	512	256	768

3.4.5.3 Model PAT-CNN

Selain tiga model *baseline* TextCNN dengan variasi kompleksitas yang dibedakan berdasarkan jumlah parameter, dimensi *embedding*, dan konfigurasi channel, penelitian ini juga mengembangkan arsitektur novel bernama *Pyramidal Attention* TextCNN (PAT-CNN). Model PAT-CNN merupakan pengembangan yang lebih *sophisticated* dan *advanced* dari ketiga model *baseline*, dimana fokus utama bukan hanya pada peningkatan kapasitas parameter, melainkan pada inovasi arsitektural yang menggabungkan *multiple advanced mechanisms* secara sinergis. Berbeda dengan model *baseline* yang menggunakan pendekatan konvensional dengan *uniform channel distribution* dan *single pooling strategy*, PAT-CNN mengintegrasikan konsep pyr*amidal feature extraction* dengan *channel-wise attention mechanism* untuk menghasilkan representasi yang lebih ekspresif dan adaptif. Model PAT-CNN memiliki inovasi baru, seperti:

1. *Pyramidal Feature Extraction*: 3 tingkat konvolusi dengan *channel* yang meningkat.
2. *Channel-wise Attention*: *Attention mechanism* untuk setiap *channel* secara terpisah.
3. *Multi-Scale Global Pooling*: Kombinasi max, avg, dan ad*aptive pooling*.
4. *Feature Fusion* dengan *Gate Mechanism*: Learnable gate untuk menggabungkan features.
5. *Dropout* yang *Adaptive*: *Dropout rate* yang berbeda untuk setiap layer.



Gambar 3.5 Ilustrasi Arsitektur Diagram PAT-CNN

Tahap pertama adalah *embedding layer* yang mengubah token input menjadi *dense vector* dengan dimensi 256. Selanjutnya, model mengimplementasikan *pyramidal convolutional layers* dengan tiga tingkat paralel: Conv1D pertama menggunakan 64 channel dengan kernel size 3, Conv1D kedua menggunakan 96 channel dengan kernel size 4, dan Conv1D ketiga menggunakan 128 channel dengan kernel size 5. Desain *pyramidal* ini memungkinkan model untuk menangkap pola n-gram dengan kompleksitas yang meningkat secara bertahap.

Setiap *convolutional layer* dilengkapi dengan cha*nnel-wise attention mechanism* yang terinspirasi dari SE-Block. *Attention module* menggunakan *adaptive average pooling* diikuti dengan dua *fully connected layer* dengan rasio reduksi 4:1 untuk menghasilkan *attention weights* yang kemudian dikalikan *element-wise* dengan *feature maps*. Hal ini memungkinkan model untuk secara adaptif memberikan bobot yang berbeda pada setiap channel berdasarkan relevansinya.

Pada tahap *pooling*, model mengimplementasikan *multi-scale global pooling* yang menggabungkan *max pooling*, *average pooling*, dan *adaptive average pooling* dengan *learnable weights*. Kombinasi ini memberikan representasi yang lebih *robust* dibandingkan *single pooling strategy*. *Feature* dari ketiga *branch* kemudian dikoncatenasi menghasilkan vektor 288 dimensi yang selanjutnya diproses melalui *feature fusion gate mechanism*.

Gate mechanism menggunakan dua *fully connected layer* dengan aktivasi sigmoid untuk menghasilkan *gate values* yang mengontrol kontribusi setiap *feature* secara selektif. *Feature* yang sudah di *gate* kemudian dikompresi dari 288 dimensi menjadi 128 dimensi untuk efisiensi komputasi. Tahap akhir adalah *classification head* dengan dua

fully connected layer (128→64→num classes) yang dilengkapi *dropout regularization* untuk mencegah *overfitting*.

Model PAT-CNN memiliki total parameter sekitar 1.85 juta dengan ukuran file 7.1 MB. Arsitektur ini dirancang untuk memberikan *trade-off* optimal 49 antara kompleksitas model dan kemampuan representasi, dengan fokus pada kemampuan untuk menangkap pola linguistik *multi-scale* melalui mekanisme *attention* yang adaptif.

3.5 Ilustrasi Perhitungan

Tahapan ini bertujuan untuk menggambarkan proses perhitungan pada model yang digunakan. Ilustrasi dilakukan untuk memberikan pemahaman mengenai bagaimana data input diproses melalui beberapa lapisan dalam TextCNN.

3.5.1 *Word Embedding Layer*

Tahapan pertama adalah penggunaan word embedding layer, layer ini bertugas mengubah setiap token menjadi vektor berdimensi tetap. Untuk memperjelas proses perhitungan pada layer ini, berikut ini merupakan contoh embedding dengan 300 dimensi.

Tokenisasi: [“mantap”, “tempat”, “perbelanjaan”, “yang”, “sangat”, “praktis”]
Tokenized indices: [10, 23, 35, 47, 51, 66]
Ukuran *embedding*: 300

Contoh 3 kata pertama:
“mantap” → [0.2, -0.1, 0.3, ..., 0.05]
“tempat” → [-0.2, 0.1, 0.0, ..., -0.1]
“perbelanjaan” → [0.0, 0.2, -0.1, ..., 0.3]

Output *embedding layer* lalu digabungkan menjadi sebuah matriks. Sebagai contoh, jika sebuah kalimat terdiri dari enam kata dan setiap kata

direpresentasikan oleh vektor berdimensi 300, maka lapisan *embedding* akan menghasilkan matriks dengan enam baris dan 300 kolom.

3.5.2 *Convolutional Layer*

Setelah proses *embedding*, data selanjutnya diproses oleh beberapa *convolutional layer* satu dimensi dengan variasi ukuran kernel (filter size). Dalam hal ini, digunakan tiga buah kernel dengan ukuran 3, 4, dan 5. Masing-masing layer konvolusi memiliki 100 buah filter, sehingga memungkinkan model untuk secara paralel mengekstrak n-gram *features* yang berbeda.

Untuk kernel size 3, satu *window* mencakup 3 kata.
Ukuran filter = 300 × 3 = 900
Misal,
kernel size = 3
bias (b) = 0.1

Gabungkan *embedding* ketiga kata menjadi vektor 900, lalu lakukan operasi konvolusi menggunakan (Persamaan 2.3.

Window 1 = $\sum(W \odot X) + b = (w_1 \times x_1 + w_2 \times x_2 + \ldots + w_{900} \times x_{900}) + b = 0.6$

Window 2 = $\sum(W \odot X) + b = (w_1 \times x_1 + w_2 \times x_2 + \ldots + w_{900} \times x_{900}) + b = -0.4$

Window 3 = $\sum(W \odot X) + b = (w_1 \times x_1 + w_2 \times x_2 + \ldots + w_{900} \times x_{900}) + b = 0.2$

Window 4 = $\sum(W \odot X) + b = (w_1 \times x_1 + w_2 \times x_2 + \ldots + w_{900} \times x_{900}) + b = 0.1$

3.5.3 *ReLU Activation*

Setelah mendapatkan nilai dari operasi konvolusi, setiap nilai negatif akan diubah menjadi 0 pada tahap ini.

[0.6, -0.4, 0.2, 0.1] → [0.6, 0.0, 0.2, 0.1]

3.5.4 *Max Pooling*

Setelah semua nilai berbentuk positif, langkah selanjutnya mencari nilai maksimum menggunakan (Persamaan 2.5.

$$MaxPool(x) = max(x_1, x_2, \ldots, x_n) = max(0.6, 0.0, 0.1) = 0.6$$

3.5.5 *Concatenate Features*

Lakukan tahap-tahap sebelumnya untuk kernel 4 dan kernel 5, misal di dapatkan hasil akhir :

kernel 3 → 0.6

kernel 4 → 0.4

kernel 5 → 0.5

Pada tahap ini penggabungan menjadi 1 vektor seperti (Persamaan 2.6.

$$f = [0.6, 0.4, 0.5] \in \mathbb{R}^3$$

3.5.6 *Dropout*

Dropout acak menyetel sebagian elemen ke nol selama *training*. Misal dropout 0.2 (20%) elemen akan dinolkan acak untuk mencegah *overfitting*.

$$f^{drop} = [0.6, 0.0, 0.5]$$

3.5.7 *Fully Connected Layer*

Input: 3 dimensi vektor dari hasil dropout

FC layer memiliki bobot $w = [w1, w2, w3]$ dan bias b.

Misal:

$w = [0.3, 0.2, 0.4]$

$b = -0.1$

Menghitung output logit menggunakan (Persamaan 2.8, output ini berarti nilai mentah yang digunakan sebagai input untuk perhitungan selanjutnya .

$$z = w \cdot f^{drop} + b = (0.3)\,(0.6) + (0.2)\,(0.0) + (0.4)\,(0.5) - 0.1$$
$$z = 0.18 + 0 + 0.20 - 0.1 = 0.28$$

Menghitung nilai sigmoid (klasifikasi biner) menggunakan (Persamaan 2.9.

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{1}{1+e^{-0.28}} \approx \frac{1}{1+0.756} \approx 0.569$$

Karena nilai sigmoid sebesar 0.569 lebih besar dari 0.5, maka hasil prediksi masuk ke kelas 1, sehingga sentimen diklasifikasikan sebagai positif.

3.6 Rancangan Pengujian

Untuk mengevaluasi performa model, digunakan metode *k-fold cross validation* untuk mengurangi risiko *overfitting* dan memberikan gambaran performa model yang lebih stabil. Dalam penelitian ini, digunakan skema 5 *fold*, yang berarti seluruh dataset dibagi secara acak menjadi lima bagian *fold* yang kurang lebih seimbang. Nilai performa model dihitung di setiap iterasi dan dirata-ratakan untuk mendapatkan hasil akhir. Setelah model selesai melakukan prediksi pada data uji di tiap *fold*, hasil klasifikasi dibandingkan dengan label sebenarnya dan disusun ke dalam *confusion matrix*. Sebagai ilustrasi, hasil prediksi label oleh model dapat dilihat pada Tabel 3.9.

Tabel 3.9 Ilustrasi Hasil Prediksi Model

Ulasan	Label Asli	Prediksi Model
Ulasan 1	1	1
Ulasan 2	0	1
Ulasan 3	1	0

Untuk melakukan perhitungan confusion matrix diperlukan Persamaan 2.12 hingga 2.15. Ilustrasi contoh perhitungan dari *confusion matrix* dapat dilihat pada Tabel 3.10.

Tabel 3.10 Ilustrasi Perhitungan *Confusion Matrix*

	Prediksi Positif (1)	**Prediksi Negatif (0)**
Positif (1)	1 (TP)	1 (FN)
Negatif (0)	1 (FP)	0 (TN)

Akurasi $= \frac{TP+TN}{TP+FP+TN+FN} = \frac{1+0}{1+0+1+1} = \frac{1}{3} \approx 0.33$ atau 33%

Presisi $= \frac{TP}{TP+FP} = \frac{1}{1+1} = \frac{1}{2} = 0.5$ atau 50%

Recall $= \frac{TP}{TP+FN} = \frac{1}{1+1} = \frac{1}{2} = 0.5$ atau 50%

F1 Score $= 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = 2 \cdot \frac{0.5 \cdot 0.5}{0.5 + 0.5} = 2 \cdot \frac{0.25}{1} = 50\%$

Nilai akurasi, presisi, *recall*, dan *f1-score* memiliki perbedaan makna karena masing-masing metrik mengukur aspek kinerja model yang berbeda. Seperti yang terlihat pada perhitungan diatas, akurasi hanya mencapai 33% karena hanya satu dari tiga prediksi yang benar. Presisi yang bernilai 50% mengindikasikan bahwa dari dua prediksi positif yang dihasilkan model, hanya satu yang benar-benar sesuai dengan label aslinya, artinya separuh dari prediksi positif adalah salah. *Recall* yang juga bernilai 50% berarti dari dua data yang sebenarnya positif, hanya satu yang berhasil dikenali oleh model, atau setengah dari data positif terlewatkan. Sementara itu, *f1-score* yang juga bernilai 50% merupakan rata-rata harmonis dari presisi dan *recall*, dan digunakan untuk menggambarkan keseimbangan antara kemampuan model dalam memprediksi positif dengan benar dan menangkap seluruh data positif. Jadi, meskipun presisi, *recall*, dan *f1-score*

sama-sama bernilai 50%, presisi fokus pada keakuratan prediksi positif, recall fokus pada kelengkapan deteksi kelas positif, dan *f1-score* menggambarkan keseimbangan keduanya.

BAB IV
HASIL DAN PEMBAHASAN

4.1 Pengumpulan Data

Proses pengumpulan data dalam penelitian ini berhasil diperoleh 5000 ulasan pengguna aplikasi Shopee yang bersumber dari Google Play Store melalui platform Kaggle untuk periode Juni hingga September 2024 melalui proses *web scraping* oleh penyedia. Distribusi jumlah ulasan berdasarkan rating bintang dapat dilihat pada Tabel 4.1 Distribusi Rating Ulasan Aplikasi Shopee yang menunjukkan dominasi ulasan dengan rating 5 bintang.

Tabel 4.1 Distribusi Rating Ulasan Aplikasi Shopee

Rating	Jumlah Ulasan	Persentase
1	756	15.12%
2	158	3.16%
3	212	4.24%
4	380	7.60%
5	3494	69.88%
Total	**5000**	**100%**

4.2 Pengembangan Model

Tahapan ini membahas proses pengembangan model TextCNN yang digunakan untuk analisis sentimen. Proses ini meliputi preprocessing data, pembentukan arsitektur model, penentuan hyperparameter, serta implementasi skema validasi silang untuk evaluasi performa. Seluruh langkah tersebut dirancang untuk memastikan model mampu mengklasifikasikan sentimen secara optimal berdasarkan data ulasan yang tersedia.

4.2.1 Input Dataset

Tahap input dataset diimplementasikan melalui kelas ShopeeComment yang merupakan turunan dari torch.utils.data.Dataset. Kelas ini berfungsi komponen utama dalam mempersiapkan data mentah menjadi format yang sesuai untuk pelatihan model TextCNN. *Source code* proses input dataset dapat dilihat pada Kode 4.1.

Kode 4.1 Input Dataset

```
file_path = "dataset.xlsx",

def load_data(self):
    self.df = pd.read_excel(self.file_path) # Load the dataset excel
    self.df.columns = ['userName', 'rating', 'timestamp', 'comment'] # Rename
    columns
    self.df = self.df.dropna(subset=['comment', 'rating']) # Drop rows with
    NaN in 'comment' and 'rating' column
    self.df['rating'] = self.df['rating'].astype(int) # Convert rating to int
    self.df = self.df[(self.df['rating'] ≥ 1) & (self.df['rating'] ≤ 5)]
    # Filter rating between 1 and 5

def __init__(self, file_path="dataset.xlsx", ...):
    # ...existing code...
    self.load_data() # Memanggil fungsi load data
    self.setup_folds() # Setup cross-validation folds
    self.setup_indices() # Setup indices untuk training/validation
```

Pada Kode 4.1 proses input dilakukan melalui fungsi load_data() yang melakukan ekstraksi data dari file Excel dataset.xlsx menggunakan *library* pandas, diikuti dengan standarisasi struktur data melalui penamaan ulang kolom menjadi 'userName', 'rating', 'timestamp', dan 'comment' untuk memastikan konsistensi format. Proses pembersihan data diterapkan dengan menghilangkan baris yang mengandung nilai kosong (NaN) pada kolom 'comment' dan 'rating', serta melakukan konversi tipe data rating menjadi integer dan *filtering* dalam rentang 1–5 untuk memastikan validitas data. Integrasi ketiga tahapan ini diatur melalui konstruktor init_() yang secara berurutan memanggil load_data() untuk inisialisasi dataset, setup_folds() untuk

pengaturan *cross-validation*, dan setup_indices() untuk penentuan subset data.

4.2.2 Data Labeling

Setelah dataset berhasil dipanggil, langkah berikutnya adalah melakukan proses pelabelan data. Proses implementasi mekanisme labeling dapat dilihat pada potongan Kode 4.2.

Kode 4.2 Data Labeling

```
def __getitem__(self, idx):
    # Hanya mengambil nomor indeks dari data yang akan diambil
    idx = self.indices[idx]

    # Mengambil data komentar dari rating
    komentar = str(self.df.iloc[idx]['comment'])
    rating = self.df.iloc[idx]['rating']
    # Mengubah rating menjadi label biner
    if rating > 3:
        label = 1
    else:
        label = 0
    # ... existing code ...

    data = {
        # ... existing code ...
        'labels': torch.tensor(label, dtype=torch.long),
        # ... existing code ...
    }
    return data
```

Proses data labeling dalam penelitian ini diimplementasikan melalui otomatis dari skala rating numerik menjadi klasifikasi biner sentimen dalam metode __getitem__() pada kelas ShopeeComment. Sistem labeling sejalan dengan penelitian sebelumnya [8], dimana rating dengan nilai 1-3 dikategorikan sebagai sentimen negatif (label 0) dan rating 4-5 sebagai sentimen positif (label 1). Proses konversi dilakukan secara real-time setiap kali DataLoader mengakses sampel data melalui operasi idx = self.indices[idx] untuk pemetaan indeks yang benar, diikuti dengan ekstraksi rating dari dataframe dan transformasi

menjadi torch.tensor(label, dtype=torch.long) yang kompatibel dengan *framework* PyTorch.

4.2.3 *Preprocessing Data*

Pada tahap ini dilakukan *preprocessing* data yang bertujuan untuk mempermudah pengolahan data pada proses klasifikasi, sehingga diharapkan mampu menghasilkan tingkat akurasi yang lebih optimal. *preprocessing* data dilakukan menggunakan *library* pada bahasa pemrograman *python*. Proses preprocessing data dalam penelitian ini diimplementasikan dalam metode *'preprocess_text()'* yang mencakup enam tahapan utama untuk mempersiapkan data teks mentah menjadi format yang optimal untuk pelatihan model.

4.2.3.1 *Case Folding*

Tahap *case folding* merupakan langkah awal dalam *preprocessing* teks yang berfungsi untuk menyelaraskan bentuk karakter dengan mengubah seluruh teks menjadi huruf kecil (*lowercase*). Proses ini penting agar kata yang sama dengan perbedaan kapitalisasi, seperti "Bagus", "BAGUS", dan "bagus", dapat dikenali sebagai satu entitas yang sama. Proses ini diimplementasikan melalui metode *'preprocess_text()'* yang dapat dilihat pada Kode 4.3, sebagai bagian dari alur *preprocessing* data.

Kode 4.3 Proses *Case Folding*

```
1 def preprocess_text(self, text):
2     # CASEFOLDING : konversi ke huruf kecil
3     text = text.lower()
```

Implementasi *case folding* pada Kode 4.3 menunjukkan pendekatan yang sederhana namun efektif dengan menggunakan fungsi '*lower()*' dari Python *string* yang mengkonversikan seluruh karakter

dalam variabel text menjadi huruf kecil. Implementasi ini membantu menjaga konsistensi data. Hasil dari proses case folding dapat dilihat pada Tabel 4.2

Tabel 4.2 Hasil Proses *Case Folding*

Input	***Case Folding Text***
Suka bgt karna banyak diskon mempermudah aku bgt	suka bgt karna banyak diskon mempermudah aku bgt
Shopee sekarang pengiriman nya lambat bangett	shopee sekarang pengiriman nya lambat bangett
udah gabisa ubah pengiriman, klau mau ubah pengiriman bayar	udah gabisa ubah pengiriman, klau mau ubah pengiriman bayar

4.2.3.2 *Cleansing*

Tahap *cleansing* dilakukan untuk menghilangkan elemen-elemen yang tidak relevan. Proses ini melibatkan penghapusan URL, karakter khusus, dan normalisasi spasi yang berlebihan untuk menghasilkan teks yang lebih bersih dan terstruktur. Tahap ini penting untuk mengurangi *noise* dalam teks.

Kode 4.4 Proses *Cleansing*

```
1 def preprocess_text(self, text):
2     # CLEANSING : hapus url
3     text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
4     # CLEANSING : menghapus special karakter
5     text = re.sub(r'[^\w\s]', '', text)
6     # Menghapus spasi berlebih
7     text = re.sub(r'\s+', ' ', text)
```

Implementasi proses cleansing pada Kode 4.4 menerapkan tiga operasi *regular expression* yang berjalan secara berurutan untuk membersihkan teks input. Operasi pertama mendeteksi dan menghapus URL yang diawali dengan protokol http/https atau awalan www.

Operasi kedua menghilangkan seluruh karakter non-alfanumerik dan non-spasi, sehingga hanya menyisakan huruf, angka, serta spasi yang relevan untuk analisis. Operasi ketiga menyederhanakan spasi berurutan menjadi satu spasi tunggal, sehingga format teks menjadi lebih konsisten dan mengurangi variasi yang tidak diperlukan dalam representasi data. Hasil proses *cleansing* dapat dilihat pada Tabel 4.3.

Tabel 4.3 Hasil Proses *Cleansing*

Case Folding Text	***Cleansing Text***
Suka bgt karna banyak diskon mempermudah aku bgt	suka bgt karna banyak diskon mempermudah aku bgt
Shopee sekarang pengiriman nya lambat bangett	shopee sekarang pengiriman nya lambat bangett
udah gabisa ubah pengiriman, klau mau ubah pengiriman bayar	udah gabisa ubah pengiriman, klau mau ubah pengiriman bayar

4.2.3.3 *Normalization*

Tahapan normalisasi merupakan proses standardisasi kata-kata informal, slang, dan singkatan menjadi bentuk formal yang sesuai dengan kaidah bahasa Indonesia baku. Normalisasi dilakukan dengan memanfaatkan kamus slang Indonesia yang diperoleh dari dataset theonlydo/indonesia-slang melalui platform Hugging Face yang menyediakan pemetaan komprehensif antara kata slang dan bentuk formalnya. Selain itu, beberapa kata tambahan juga disertakan secara manual untuk menyelesaikan dengan dataset yang digunakan.

Kode 4.5 Proses *Normalization*

```
def load_normalization_dict(json_path=None):
    try:
        dataset = load_dataset("theonlydo/indonesia-slang", split="train")
        normalization_dict = {row['slang']: row['formal'] for row in
        dataset}
        return normalization_dict
    except Exception as e:
        print(f"Gagal load kamus dari HuggingFace: {e}")
        return {}

def __init__(self, ...):
    # ...existing code...
    self.normalization_dict = load_normalization_dict(normalization_file)

def preprocess_text(self, text):
    # Normalization : dengan kamus dari file JSON
    words = [self.normalization_dict.get(word, word) for word in words
```

Implementasi normalisasi pada Kode 4.5 terdiri dari tiga komponen utama yang bekerja secara terintegrasi untuk memastikan standardisasi bahasa. Fungsi_load_normalization_dict() bertugas memuat dataset slang dari Hugging Face dan mengkonversinya menjadi *dictionary* dengan struktur *key-value* yang memetakan kata slang ke bentuk formalnya. Proses inisialisasi kamus dilakukan pada saat dipanggil melalui self.normalization_dict = load_normalization_dict(). Penerapan normalisasi dalam preprocess_text() menggunakan *list comprehension* dengan metode dict.get(word, word) yang mencari setiap kata dalam kamus normalisasi dan menggantikannya dengan bentuk formal jika ditemukan, atau mempertahankan kata asli jika tidak ada dalam kamus. Hasil dari proses *normalization* dapat dilihat pada Tabel 4.4.

Tabel 4.4 Hasil Proses *Normalization*

Cleansing Text	***Normalization Text***
Suka bgt karna banyak diskon mempermudah aku bgt	suka banget karena banyak diskon mempermudah aku banget

Cleansing Text	*Normalization Text*
Shopee sekarang pengiriman nya lambat bangett	shopee sekarang pengiriman nya lambat banget
udah gabisa ubah pengiriman, klau mau ubah pengiriman bayar	udah ga bisa ubah pengiriman, kalau mau ubah pengiriman bayar

4.2.3.4 *Tokenizing*

Pada tahap ini, teks dipecah menjadi token untuk mempermudah analisis. Tokenisasi dilakukan dengan dua pendekatan, yaitu berbasis NLTK untuk prapemrosesan tradisional seperti normalisasi dan penghapusan *stopwords*, serta IndoBERT tokenizer untuk menghasilkan representasi numerik yang sesuai dengan arsitektur transformer.

Kode 4.6 Proses *Tokenizing*

```
def preprocess_text(self, text):
    # Tokenisasi
    words = nltk.word_tokenize(text)

def __getitem__(self, idx):
    ...existing code...
    # Tokenisasi
    encoding = self.tokenizer(
        comment_processed,
        add_special_tokens=True,
        max_length=128,
        padding='max_length',
        truncation=True,
        return_attention_mask=True,
        return_tensors="pt",
    )
```

Implementasi tokenisasi pada Kode 4.6 menunjukkan arsitektur dualphase yang mengoptimalkan proses *preprocessing* dan representasi data. Langkah awal dalam preprocess_text() menggunakan nltk.word_tokenize() yang memecah teks menjadi *array* kata-kata terpisah dengan mempertahankan struktur bahasa Indonesia,

memungkinkan operasi *preprocessing* selanjutnya seperti normalisasi dan *stopwords removal*. Pada __getitem__() menerapkan IndoBERT *tokenizer* dengan konfigurasi add_special_tokens=True untuk menambahkan token khusus [CLS] dan [SEP], max length=128 untuk standardisasi panjang input, padding='max length' dan truncation=True untuk menangani variabilitas panjang teks, serta return_tensors="pt" untuk menghasilkan PyTorch tensor yang siap digunakan model. Pendekatan ini memastikan bahwa *preprocessing* tradisional dapat berjalan optimal pada representasi kata yang mudah dipahami, sementara model *neural network* menerima input berupa token ID dan *attention mask* yang sesuai dengan arsitektur transformer sehingga mencapai keseimbangan. Hasil dari proses *normalization* dapat dilihat pada Tabel 4.5.

Tabel 4.5 Hasil Proses *Tokenizing*

Normalization Text	***Tokenizing Text***
suka banget karena banyak diskon mempermudah aku banget	["suka", "banget", "karena", "banyak", "diskon", "mempermudah", "saya", "banget"]
shopee sekarang pengiriman nya lambat banget	["shopee", "sekarang", "pengiriman", "nya", "lambat", "banget"]
udah ga bisa ubah pengiriman, kalau mau ubah pengiriman bayar	["udah", "ga", "bisa", "ubah", "pengiriman", "kalau", "mau", "ubah", "pengiriman", "bayar"]

4.2.3.5 *Stopwords Removal*

Tahapan *stopwords removal* untuk menghapus kata-kata yang tidak mempunyai makna penting (*stopwords*) dalam analisis sentimen,

seperti kata hubung, kata ganti, dan partikel yang tidak memberikan informasi signifikan. Implementasi *stopwords removal* menggunakan korpus bahasa Indonesia dari NLTK yang telah dikurasi secara khusus untuk menangani karakteristik bahasa Indonesia.

Kode 4.7 Proses *Stopwords Removal*

```
# Inisialisasi Stopwords
try:
    nltk.data.find('tokenizers/punkt')
except LookupError:
    nltk.download('punkt')

try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    nltk.download('stopwords')

INDONESIAN_STOPWORDS = set(stopwords.words('indonesian'))
def preprocess_text(self, text):
    # STOPWORDS : menghapus kata yang tidak penting
    words = [word for word in words if word not in INDONESIAN_STOPWORDS]
```

Implementasi *stopwords removal* pada Kode 4.7 terdiri dari dua tahapan utama yang memastikan ketersediaan dan efisiensi penggunaan *korpus stopwords* bahasa Indonesia. Tahapan pembentukan INDONESIAN STOPWORDS sebagai set data struktur yang mengoptimalkan operasi pencarian dengan kompleksitas. Proses penghapusan *stopwords* dalam preprocess_text() menggunakan *list comprehension* [word for word in words if word not in INDONESIAN STOPWORDS] yang secara efisien memfilter setiap kata dalam *array* hasil tokenisasi, mempertahankan hanya kata-kata yang tidak terdapat dalam set *stopwords* Indonesia. Pendekatan ini mengurangi *noise* dalam data input dan memungkinkan model TextCNN untuk lebih fokus pada kata-kata yang memiliki nilai semantik tinggi dalam menentukan polaritas sentimen, sekaligus mengurangi beban komputasi melalui pengurangan dimensionalitas fitur yang signifikan tanpa mengorbankan

informasi penting untuk klasifikasi. Hasil dari proses *normalization* dapat dilihat pada Tabel 4.6.

Tabel 4.6 Hasil Proses *Stopwords Removal*

Tokenizing Text	***Stopwords Removal Text***
[“suka”, “banget”, “karena”, “banyak”, “diskon”, “mempermudah”, “saya”, “banget”]	[“suka”, “banget”, “diskon”, “mempermudah”, “banget”]
[“shopee”, “sekarang”, “pengiriman”, “nya”, “lambat”, “banget”]	[“shopee”, “pengiriman”, “lambat”, “banget”]
[“udah”, “ga”, “bisa”, “ubah”, “pengiriman”, “kalau”, “mau”, “ubah”, “pengiriman”, “bayar”]	[“udah”, “ga”, “bisa”, “ubah”, “pengiriman”, “ubah”, “pengiriman”, “bayar”]

4.2.3.6 Augmentasi Teks

Tahapan augmentasi data merupakan teknik regularisasi yang bertujuan untuk memperluas variabilitas dataset pelatihan melalui modifikasi terstruktur pada teks input, sehingga meningkatkan kemampuan generalisasi model dan mengurangi risiko *overfitting*. Augmentasi dilakukan secara berurutan, di mana setiap jenis transformasi diterapkan satu per satu pada teks yang sama. Proses ini mencakup lima jenis transformasi utama, yaitu *replace phrases, synonym replacement, random typo, random swap,* dan *random delete.*

Kode 4.8 Proses Augmentasi Teks

```
def augment_text(self, text):
    # Phrase replace
    if random.random() < self.phrase_prob:
        for phrase, replacements in
  self.augmentasi_data.get("replace_phrases", {}).items():
            if phrase in text:
                text = text.replace(phrase, random.choice(replacements))
    # Synonym replacement
    words = text.split()
    if random.random() < self.synonym_prob:
        for i, word in enumerate(words):
            if word in self.augmentasi_data.get("synonyms", {}):
                words[i] = random.choice(self.augmentasi_data["synonyms"]
                           [word])
    text = ' '.join(words)
    tokens = word_tokenize(text)
    tokens = [t for t in tokens if t not in INDONESIAN_STOPWORDS]
    text = ' '.join(tokens)
    # Typo
    if random.random() < self.typo_prob:
        text = self.random_typo(text)
    # Swap
    if random.random() < self.swap_prob:
        text = self.random_swap(text)
    # Delete
    if random.random() < self.delete_prob:
        text = self.random_delete(text)

    return text

def preprocess_text(self, text):
    # Augmentasi : typo, swap, delete
    text = self.augment_text(text)
```

Implementasi augmentasi data pada Kode 4.8 menunjukkan pendekatan multi-layer yang menggabungkan perubahan makna dan struktur kalimat untuk menghasilkan variasi data yang realistis namun tetap mempertahankan label sentimen yang konsisten. Proses dimulai dengan *phrase replacement* yang mengganti frasa tertentu dengan sinonim berdasarkan kamus yang telah disusun, diikuti dengan synonym replacement pada level kata individual menggunakan probabilitas self.synonym_prob untuk mengatur frekuensi penggantian. Tahapan tengah melibatkan tokenisasi dan penghapusan *stopwords* untuk membersihkan hasil transformasi sebelum menerapkan augmentasi yang lebih intensif seperti random_typo() untuk simulasi

kesalahan ketik, random_swap() untuk pertukaran posisi kata, dan random_delete() untuk penghapusan kata acak. Hasil dari proses augmentasi teks dapat dilihat pada Tabel 4.7.

Tabel 4.7 Hasil Proses Augmentasi Data

Sebelum	Sesudah
suka banget diskon mempermudah banget	sekali diskon mempermudah sekali
shopee pengiriman lambat banget	delivery lambat sekali
udah ga bisa ubah pengiriman ubah pengiriman bayar	delivery ubah pengiriman bayar

4.2.4 Split Data

Pada tahap ini dilakukan proses split data untuk memastikan evaluasi model melalui pembagian data yang seimbang. Implementasi split data menggunakan teknik *Stratified K-Fold Cross-Validation* dengan k=5, sehingga setiap subset pelatihan dan validasi memiliki representasi yang seimbang.

Kode 4.9 Pembuatan *Folds* dengan *Stratified K-Fold*

```python
from sklearn.model_selection import StratifiedKFold

def create_folds(self):
    print(f"Membuat 5-folds cross-validation dengan random state {self.random_state}")
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=self.random_state)

    fold_indices = {}
    for fold, (train_idx, val_idx) in enumerate(skf.split(self.df, self.df['rating'])):
        fold_indices[f"fold_{fold}"] = {
            "train_indices": train_idx.tolist(),
            "val_indices": val_idx.tolist()
        }

    # Save fold indices to JSON file
    with open(self.folds_file, 'w') as f:
        json.dump({
            'fold_indices' : fold_indices,
            'n_samples' : len(self.df),
            'n_folds': 5,
            'random_state': self.random_state
        }, f)

    self.folds = fold_indices
```

Pada Kode 4.9 digunakan fungsi StratifiedKFold(nsplits=5, shuffle=True, randomstate=2025) menciptakan objek pembagi dengan parameter n splits=5, pembagian data pada setiap iterasi secara implisit menghasilkan data rasio 80% data pelatihan dan 20% data validasi seperti Tabel 4.8.

Tabel 4.8 Split Data dengan *Stratified K-Fold Cross Validation*

Iterasi	**Fold**					***Train Data***	***Validation Data***
	0	**1**	**2**	**3**	**4**		
1	1000	1000	1000	1000	1000	4000	1000
2	1000	1000	1000	1000	1000	4000	1000
3	1000	1000	1000	1000	1000	4000	1000
4	1000	1000	1000	1000	1000	4000	1000
5	1000	1000	1000	1000	1000	4000	1000

Pada setiap iterasi, satu *fold* digunakan sebagai data validasi, sedangkan empat *fold* lainnya digunakan sebagai data pelatihan seperti pada Tabel 4.8. Setiap kolom berwarna merah menandakan *fold* yang digunakan sebagai data validasi pada iterasi ke-n. Pada iterasi pertama, *fold* 1 digunakan sebagai data validasi dan *fold* 2 hingga *fold* 5 sebagai data pelatihan. Proses ini berlanjut hingga seluruh *fold* memperoleh giliran sebagai data validasi sebanyak satu kali. Adapun jumlah data latih yang diperoleh adalah 4000, sementara jumlah data validasi sebanyak 1000.

Proses iterasi menggunakan skf.split(self.df, self.df['rating']) menghasilkan pasangan indeks pelatihan dan validasi untuk setiap *fold* yang kemudian disimpan dalam *dictionary* terstruktur dengan format fold_indices[f"foldfold"] yang memisahkan *train indices* dan *val indices*. Mekanisme penyimpanan melalui json.dump() menghasilkan

file JSON yang berisi meta data seperti jumlah sampel, jumlah *fold*, dan *random state*. Hasil pembagian data dengan *StratifiedK-Fold* selanjutnya diatur dan disimpan melalui proses *setup folds* seperti pada Kode 4.10.

Kode 4.10 *Setup Folds*

```
def setup_folds(self):
    # Check if folds file exists
    if os.path.exists(self.folds_file):
        self.load_folds()
    else: # Create folds if file does not exist
        self.create_folds()
        self.save_folds()
```

Implementasi *setup folds* pada Kode 4.10 menggunakan logika pengecekan file JSON untuk menentukan apakah konfigurasi *fold* perlu dimuat melalui self.load_folds() atau dibuat baru dengan self.create_folds() dan disimpan menggunakan self.save_folds(). Proses ini membuat pembagian data tetap konsisten dan dapat diulang, serta memberikan fleksibilitas untuk membuat *fold* baru jika diperlukan. Tahap berikutnya adalah memuat kembali hasil pembagian tersebut melalui mekanisme *load folds* seperti Kode 4.11.

Kode 4.11 *Load Folds*

```
def load_folds(self):
    with open(self.folds_file, 'r') as f:
        folds_data = json.load(f)

    self.folds_indices = folds_data['fold_indices']
    self.folds = self.folds_indices
    print(f"Menggunakan {folds_data['n_folds']} folds dengan {folds_data['n_samples']}
     samples dataset")
```

Implementasi *load folds* pada Kode 4.11 menunjukkan proses pembacaan kembali konfigurasi pembagian data dari file yang telah disimpan. Pembacaan data menggunakan context manager with open(self.folds_file, ’r’) untuk membaca file JSON secara aman dan json.load(f) untuk mengkonversi data JSON menjadi struktur

dictionary Python yang berisi informasi lengkap tentang pembagian *fold*. Ekstraksi indeks *fold* dilakukan melalui self.folds_indices = folds_data[’fold indices’] yang mengambil *mapping* antara setiap *fold* dengan indeks *training* dan *validation* yang telah ditentukan sebelumnya, kemudian disimpan dalam atribut self.folds untuk akses yang lebih mudah dalam proses seleksi data. Setelah konfigurasi *fold* berhasil dimuat melalui *load folds*, langkah berikutnya adalah menetapkan indeks data ke dalam subset *train* dan *validation* melalui proses *setup indices* seperti Kode 4.12.

Kode 4.12 *Setup Indices* untuk *Train* atau *Validation*

```
def setup_indices(self):
    # Mempersiapkan indices untuk data yang akan di training
    fold_key = f"fold_{self.fold}"

    if self.split == "train" :
        self.indices = self.folds[fold_key]['train_indices']
    else:
        self.indices = self.folds[fold_key]['val_indices']
```

Implementasi *setup indices* pada Kode 4.12 menunjukkan mekanisme *runtime selection* yang memungkinkan pemilihan subset data secara fleksibel berdasarkan jenis operasi dan *fold* yang sedang digunakan dalam proses *cross validation*. Sistem menggunakan string formatting fold_key = f”fold self.fold” untuk mengonstruksi kunci akses yang sesuai dengan *fold* yang sedang aktif, kemudian menerapkan *conditional logic* berdasarkan parameter self.split untuk menentukan apakah *instance* dataset akan menggunakan indeks *training* atau *validation*. Pemilihan indeks dilakukan melalui akses *dictionary* self.folds[fold_key] [’train_indices’] atau self.folds[fold_key] [’val_indices’] yang mengambil *array* indeks yang telah dipredefinisi dalam proses pembagian *fold* sebelumnya, kemudian disimpan dalam

atribut self.indices yang akan digunakan oleh metode __getitem__() untuk mengakses sampel data yang tepat.

4.2.5 Pemodelan

Pada tahap ini dilakukan proses pemodelan menggunakan model TextCNN dengan berbagai parameter yang di variasikan. Model dilatih dengan data yang telah melalui tahap *preprocessing*. Model dilatih dan diuji secara bergantian pada beberapa bagian data sehingga performa yang diperoleh tidak bergantung pada satu kali pembagian data.

4.2.5.1 Model TextCNN Ringan

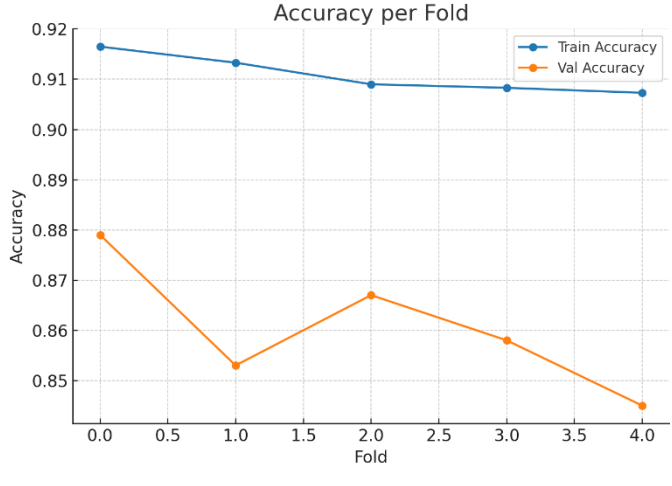
Arsitektur model ini dirancang dengan parameter yang minimal namun efektif untuk menangkap karakteristik sentimen dalam teks bahasa Indonesia dengan total parameter yang relatif kecil seperti pada Kode 4.13.

Kode 4.13 Model TextCNN Ringan

```
class TextCNN(nn.Module):
    def __init__(self, vocab_size, embed_dim=100, num_classes=2, do=0.2):
        super(TextCNN, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=0)
        # Layer CNN yang digunakan:
        self.conv1 = nn.Conv1d(embed_dim, 50, kernel_size=3)
        self.conv2 = nn.Conv1d(embed_dim, 50, kernel_size=4)
        self.conv3 = nn.Conv1d(embed_dim, 50, kernel_size=5)
        self.dropout = nn.Dropout(do)
        self.fc = nn.Linear(150, num_classes)

    def forward(self, x):
        x = self.embedding(x).permute(0, 2, 1)
        x1 = F.relu(self.conv1(x)).max(dim=2)[0]
        x2 = F.relu(self.conv2(x)).max(dim=2)[0]
        x3 = F.relu(self.conv3(x)).max(dim=2)[0]
        x = torch.cat((x1, x2, x3), dim=1)
        x = self.dropout(x)
        return self.fc(x)
```

Pada Kode 4.13 menunjukkan bahwa model menggunakan *embedding layer* dengan dimensi 100 (embed_dim=100) yang mengkonversi token ID menjadi representasi vektor *dense* dengan

padding_idx=0 untuk menangani padding token, diikuti dengan tiga paralel *convolutional layer* (conv1, conv2, conv3) yang menggunakan *kernel size* berbeda (3, 4, 5) dengan 50 filter masing-masing untuk menangkap n-gram pattern dengan panjang yang bervariasi. Proses *forward propagation* melibatkan transformasi *embedding* melalui permute(0, 2, 1) untuk menyesuaikan dimensi *input convolution*, aplikasi fungsi aktivasi ReLU dan *max-pooling* global max(dim=2)[0] pada setiap *convolution output*, *concatenation* hasil *pooling* menggunakan torch.cat() yang menghasilkan *feature vector* berukuran 150, dan regularisasi melalui *dropout* layer sebelum klasifikasi final menggunakan *fully connected layer*. Pada Kode 4.13 menampilkan arsitektur model yang digunakan, sedangkan hasil pelatihan model dapat dilihat pada Gambar 4.1, yang menampilkan perbandingan nilai *train accuracy* dan *validation accuracy* pada setiap *fold.*



Gambar 4.1 Hasil Akurasi Train dan Val Model TextCNN Ringan

Pada Gambar 4.1 terlihat bahwa grafik dari nilai akurasi pelatihan konsisten lebih tinggi dibandingkan akurasi validasi. Kondisi ini merupakan hal yang sering terjadi dalam *processing text* karena

model dilatih langsung pada data pelatihan sehingga mampu mengenali pola dengan lebih baik, sedangkan data validasi digunakan untuk menguji kemampuan pemodelan menyeluruh pada data yang belum pernah dilihat. Gap antara akurasi pelatihan dan validasi yang berkisar 4-6% menunjukkan adanya tingkat *overfitting* ringan pada model ini. Hal ini dipicu oleh beberapa faktor seperti augmentasi teks pada data latih menciptakan variasi sintetis yang tidak sepenuhnya merepresentasikan data validasi, serta kapasitas model dengan 3.36M parameter cenderung menghafal pola spesifik sehingga kurang optimal dalam generalisasi, namun masih dalam batas yang dapat diterima untuk generalisasi yang baik. Pola penurunan akurasi validasi yang terjadi pada *fold* 1 dan kemudian membaik pada *fold* 2 mengindikasikan variabilitas data antar *fold* yang mempengaruhi konsistensi performa model.

4.2.5.2 Model TextCNN Sedang

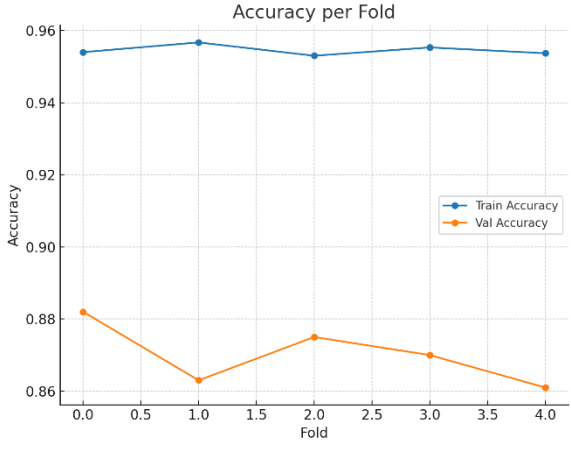
Arsitektur TextCNN model sedang merupakan pengembangan dari versi ringan dengan peningkatan kapasitas representasi melalui ekspansi dimensi embedding dan jumlah filter konvolusi untuk mengakomodasi kompleksitas yang lebih tinggi dalam analisis sentimen seperti pada Kode 4.14.

Kode 4.14 Model TextCNN Sedang

```
class TextCNN(nn.Module):
    def __init__(self, vocab_size, embed_dim=300, num_classes=2, do=0.2):
        super(TextCNN, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=0)
        self.conv1 = nn.Conv1d(embed_dim, 100, kernel_size=3)
        self.conv2 = nn.Conv1d(embed_dim, 100, kernel_size=4)
        self.conv3 = nn.Conv1d(embed_dim, 100, kernel_size=5)
        self.dropout = nn.Dropout(do)
        self.fc = nn.Linear(300, num_classes)
```

```
10
11      def forward(self, x):
12          x = self.embedding(x).permute(0, 2, 1)
13          x1 = F.relu(self.conv1(x)).max(dim=2)[0]
14          x2 = F.relu(self.conv2(x)).max(dim=2)[0]
15          x3 = F.relu(self.conv3(x)).max(dim=2)[0]
16          x = torch.cat((x1, x2, x3), dim=1)
17          x = self.dropout(x)
18          return self.fc(x)
```

Pada Kode 4.14 menunjukkan scaling up yang signifikan dengan peningkatan dimensi embedding menjadi 300 (embed dim=300) untuk representasi kata yang lebih kaya dan ekspansi jumlah filter konvolusi menjadi 100 per kernel size (3, 4, 5) yang menghasilkan total 300 feature maps. Peningkatan kapasitas ini memungkinkan model untuk menangkap pola linguistik yang lebih kompleks dan nuansa semantik yang lebih halus dalam teks ulasan, dengan *embedding layer* yang dapat merepresentasikan relasi kata dengan dimensionalitas yang lebih tinggi dan *convolutional layer* yang dapat mengekstrak lebih banyak n-gram pattern yang relevan untuk klasifikasi sentiment. Proses *forward propagation* tetap mengikuti alur yang sama dengan model ringan, namun dengan *feature vector* hasil *concatenation* yang berukuran 300 (100×3) sebelum masuk ke *fully connected layer* untuk klasifikasi final. Pada Kode 4.14 ditunjukkan arsitektur model yang digunakan, sedangkan hasil pelatihan model dapat dilihat pada Gambar 4.2, yang menampilkan perbandingan nilai *train accuracy* dan *validation accuracy* pada setiap *fold.*



Gambar 4.2 Hasil Akurasi Train dan Val Model TextCNN Sedang

Pada Gambar 4.2 ditunjukkan grafik perbandingan nilai akurasi pada data *train* dan *validation* untuk setiap *fold*. Terlihat bahwa akurasi pada data *train* cenderung stabil dengan kisaran sekitar 0.953 – 0.957, menunjukkan bahwa model mampu belajar dengan konsisten pada data latih. Sementara, akurasi pada data *validation* memiliki nilai yang relatif lebih rendah dibandingkan akurasi *train*, yaitu berada pada kisaran 0.861 – 0.882. Adanya gap 7–9% antara *train* dan *validation accuracy* pada model ini dapat disebabkan oleh kapasitas yang lebih besar (10.08M parameter) yang membuat model mampu menangkap pola kompleks dari data augmentasi, namun juga lebih rentan *overfitting* terhadap *noise* sehingga mengurangi kemampuan generalisasi ke data validasi. Karakteristik arsitektur TextCNN yang menggunakan *convolutional layers* untuk menangkap pola n-gram secara hierarkis turut memperkuat kecenderungan model mempelajari fitur yang sangat spesifik dari teks latih.

4.2.5.3 Model TextCNN Berat

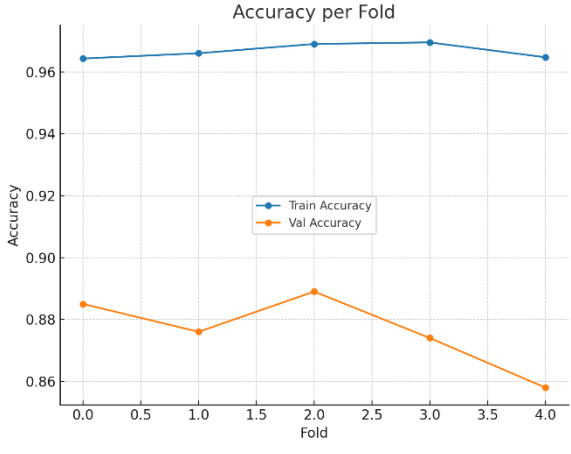
Model arsitektur TextCNN model berat merepresentasikan konfigurasi dengan kapasitas maksimal yang dirancang untuk menangani kompleksitas tinggi dalam analisis sentimen dengan mengorbankan efisiensi komputasi demi akurasi dan kemampuan representasi yang superior seperti pada Kode 4.15.

Kode 4.15 Model TextCNN Berat

```
class TextCNN(nn.Module):
    def __init__(self, vocab_size, embed_dim=512, num_classes=2, do=0.2):
        super(TextCNN, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=0)
        self.conv1 = nn.Conv1d(embed_dim, 256, kernel_size=3)
        self.conv2 = nn.Conv1d(embed_dim, 256, kernel_size=4)
        self.conv3 = nn.Conv1d(embed_dim, 256, kernel_size=5)
        self.dropout = nn.Dropout(do)
        self.fc = nn.Linear(768, num_classes)

    def forward(self, x):
        x = self.embedding(x).permute(0, 2, 1)
        x1 = F.relu(self.conv1(x)).max(dim=2)[0]
        x2 = F.relu(self.conv2(x)).max(dim=2)[0]
        x3 = F.relu(self.conv3(x)).max(dim=2)[0]
        x = torch.cat((x1, x2, x3), dim=1)
        x = self.dropout(x)
        return self.fc(x)
```

Pada Kode 4.15 menunjukkan *scaling* maksimal dengan dimensi *embedding* 512 (embed.dim=512) yang memberikan representasi kata dengan ruang vektor yang sangat kaya, dan 256 filter per kernel size (3,4,5) yang menghasilkan total 768 *feature maps* untuk ekstraksi pola yang komprehensif. Peningkatan dramatis dalam jumlah parameter ini memungkinkan model untuk mempelajari representasi yang sangat detail dan menangkap interaksi semantik yang kompleks antar kata dalam konteks sentimen, dengan *embedding layer* yang dapat merepresentasikan nuansa makna yang sangat halus dan *convolutional layer* yang dapat mengidentifikasi pola n-gram yang beragam dengan granularitas tinggi. Arsitektur ini menghasilkan *feature vector* berukuran 768 (256×3) setelah proses *concatenation*, memberikan

kapasitas representasional yang sangat besar untuk *fully connected layer* dalam melakukan klasifikasi final. Pada Kode 4.15 ditunjukkan arsitektur model yang digunakan, sedangkan hasil pelatihan model dapat dilihat pada Gambar 4.3, yang menampilkan perbandingan nilai *train accuracy* dan *validation accuracy* pada setiap fold.



Gambar 4.3 Hasil Akurasi Train dan Val Model TextCNN Berat

Pada Gambar 4.3 menunjukkan bahwa grafik akurasi pada data latih konsisten sangat tinggi, berada pada kisaran 0.964 hingga 0.969, yang menandakan model mampu mempelajari pola dengan baik di setiap *fold*. Akurasi pada data validasi juga terlihat cukup stabil, berada antara 0.858 hingga 0.889, dengan capaian terbaik pada *fold* ke-2. Gap 8–11% antara *training* dan *validation accuracy* pada model berat menunjukkan bahwa kapasitas besar (17.20M parameter) membawa konsekuensi berupa risiko *overfitting* yang lebih tinggi. Dengan *embedding* 512 dan 256 filter per kernel, model mampu menghafal detail data latih termasuk pola tambahan dari hasil augmentasi, namun kesulitan mengeneralisasi ke data baru. Meski demikian, akurasi

validasi yang stabil di kisaran 85–88% menegaskan bahwa pola utama tetap berhasil ditangkap sehingga performa tetap memadai.

4.2.5.4 Model PAT-CNN

Arsitektur *Pyramidal Attention* TextCNN (PAT-CNN) merupakaninovasi arsitektur *neural network* yang menggabungkan konsep *pyramidal feature extraction* dengan mekanisme *attention* untuk meningkatkan kemampuan ekstraksi fitur *hierarchical* dalam analisis sentimen. Arsitektur ini dirancang dengan lima komponen utama yang saling terintegrasi yaitu *pyramidal convolution* dengan peningkatan channel secara bertahap, *channel-wise attention mechanism, multi-scale global pooling* dengan *learnable weights*, *feature fusion* menggunakan *gate mechanism*, dan dropout adaptif agar optimal pada setiap *layer* seperti pada Kode 4.16.

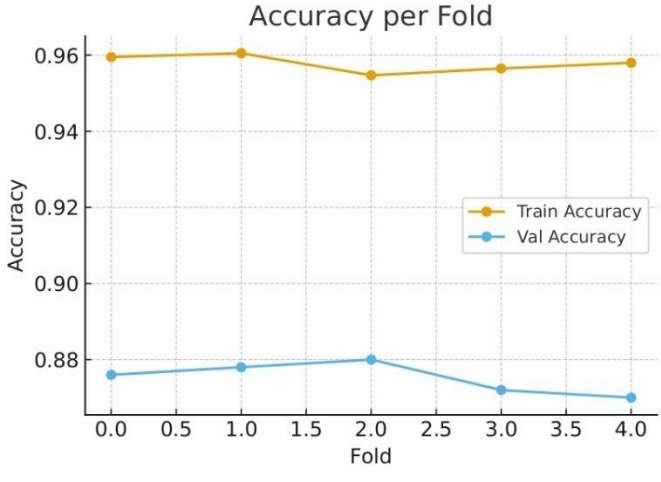
Kode 4.16 Model PAT-CNN

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class TextCNN_X(nn.Module):
    def __init__(self, vocab_size, embed_dim=256, num_classes=2, do=0.2):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=0)
        self.conv1 = nn.Conv1d(embed_dim, 64, 3, padding=1)
        self.conv2 = nn.Conv1d(embed_dim, 96, 4, padding=2)
        self.conv3 = nn.Conv1d(embed_dim, 128, 5, padding=2)
        self.bn1 = nn.BatchNorm1d(64)
        self.bn2 = nn.BatchNorm1d(96)
        self.bn3 = nn.BatchNorm1d(128)
        self.channel_attention1 = nn.Sequential(
            nn.AdaptiveAvgPool1d(1), nn.Conv1d(64, 16, 1), nn.ReLU(),
            nn.Conv1d(16, 64, 1), nn.Sigmoid())
        self.channel_attention2 = nn.Sequential(
            nn.AdaptiveAvgPool1d(1), nn.Conv1d(96, 24, 1), nn.ReLU(),
            nn.Conv1d(24, 96, 1), nn.Sigmoid())
        self.channel_attention3 = nn.Sequential(
            nn.AdaptiveAvgPool1d(1), nn.Conv1d(128, 32, 1), nn.ReLU(),
            nn.Conv1d(32, 128, 1), nn.Sigmoid())
        self.pooling_weights = nn.Parameter(torch.tensor([0.4, 0.3, 0.3]))
        self.fusion_gate = nn.Sequential(
            nn.Linear(288, 144), nn.ReLU(), nn.Linear(144, 288), nn.Sigmoid())
        self.feature_compress = nn.Sequential(
            nn.Linear(288, 128), nn.ReLU(), nn.Dropout(do * 0.5))
        self.classifier = nn.Sequential(
            nn.Linear(128, 64), nn.ReLU(), nn.Dropout(do), nn.Linear(64,
            num_classes))
```

```
    def multi_scale_pooling(self, x, w):
        maxp = F.max_pool1d(x, x.size(2)).squeeze(2)
        avgp = F.avg_pool1d(x, x.size(2)).squeeze(2)
        adap = F.adaptive_avg_pool1d(x, 1).squeeze(2)
        w = F.softmax(w, dim=0)
        return w[0]*maxp + w[1]*avgp + w[2]*adap

    def forward(self, x):
        x = self.embedding(x).permute(0, 2, 1)
        c1 = F.relu(self.bn1(self.conv1(x)))
        c2 = F.relu(self.bn2(self.conv2(x)))
        c3 = F.relu(self.bn3(self.conv3(x)))
        a1 = self.channel_attention1(c1)
        a2 = self.channel_attention2(c2)
        a3 = self.channel_attention3(c3)
        p1 = self.multi_scale_pooling(c1 * a1, self.pooling_weights)
        p2 = self.multi_scale_pooling(c2 * a2, self.pooling_weights)
        p3 = self.multi_scale_pooling(c3 * a3, self.pooling_weights)
        concat = torch.cat([p1, p2, p3], dim=1)
        gate = self.fusion_gate(concat)
        x = self.feature_compress(concat * gate)
        return self.classifier(x)
```

Pada Kode 4.16 menunjukkan bahwa arsitektur mengintegrasikan *multiple innovation* dalam *deep learning* untuk *text classification*. Komponen *pyramidal convolution* menggunakan tiga layer konvolusi dengan peningkatan channel secara progresif (64,96,128) dan *kernel size* yang berbeda (3,4,5) untuk menangkap n-gram *pattern* dengan kompleksitas yang meningkat, sementara setiap layer konvolusi dipasangkan dengan *channel-wise attention mechanism* yang menggunakan *squeeze-excitation approach* melalui AdaptiveAvgPool1d dan MLP untuk mempelajari *importance weight* setiap channel. *Multi-scale global pooling* dengan *learnable* parameter self.pooling_weights memungkinkan model untuk secara adaptif menggabungkan *max pooling*, *average pooling*, dan *adaptive pooling* berdasarkan karakteristik data, sedangkan *feature fusion gate mechanism* menggunakan *sigmoid activation* untuk mengontrol kontribusi setiap *feature* secara selektif. Arsitektur ini menghasilkan representasi yang lebih kaya dan *discriminative* dibandingkan TextCNN konvensional, dengan kemampuan untuk fokus pada fitur

yang paling relevan melalui *attention mechanism* dan mengoptimalkan kombinasi *multi-scale features* melalui *learnable pooling weights*. Pada Kode 4.16 ditunjukkan arsitektur model yang digunakan, sedangkan hasil pelatihan model dapat dilihat pada Gambar 4.4, yang menampilkan perbandingan nilai *train accuracy* dan *validation accuracy* pada setiap *fold*.



Gambar 4.4 Hasil Akurasi Train dan Val Model PAT-CNN

Pada Gambar 4.4 menunjukkan bahwa grafik akurasi pada data latih konsisten berada pada kisaran 95,4% hingga 96,0%, menandakan model mampu mempelajari pola dari data dengan baik. Sementara itu, akurasi pada data validasi berada pada kisaran 87,0% hingga 88,0%, yang lebih rendah dari akurasi pelatihan namun tetap stabil pada setiap fold. Selisih akurasi pelatihan dan validasi sebesar 7–9% menunjukkan adanya *overfitting* ringan pada arsitektur PAT-CNN yang kompleks. Mekanisme *attention* dan *feature fusion gate* yang dipadukan dengan regularisasi (*dropout* adaptif, *batch normalization*) serta augmentasi agresif diduga menciptakan bias terhadap variasi hasil augmentasi. Meski demikian, performa validasi yang stabil di seluruh fold

menegaskan kemampuan prediksi model tetap memadai dengan kontrol overfitting yang cukup efektif.

4.3 Hasil Pengujian dan Evaluasi

Pengujian performa model dilakukan untuk menilai efektivitas TextCNN dengan konfigurasi parameter yang berbeda. Proses evaluasi menerapkan metode *Stratified K-Fold Cross Validation* sebanyak 5 *fold* pada data latih, dengan penilaian berfokus pada empat metrik utama yaitu akurasi, presisi, *recall*, serta *f1-score*. Analisis ini bertujuan untuk mengetahui sejauh mana model mampu melakukan klasifikasi secara konsisten.

4.3.1 Model TextCNN Ringan

Untuk mengevaluasi performa model TextCNN pada setiap fold, digunakan *confusion matrix* yang berisi jumlah prediksi benar maupun salah dari kelas positif dan negatif. Matriks ini merepresentasikan nilai *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), dan *False Negative* (FN).

Tabel 4.9 Hasil Evaluasi Model TextCNN Ringan

Fold	**Confusion Matrix**				**Metrik Evaluasi**			
	TP	TN	FP	FN	Acc	Pre	Rec	F1
0	730	149	77	44	0.879	0.9064	0.9432	0.9235
1	741	122	113	34	0.853	0.8677	0.9561	0.9098
2	740	127	98	35	0.867	0.8831	0.9548	0.9175
3	737	121	104	38	0.858	0.8763	0.9510	0.9121
4	715	130	95	60	0.845	0.8827	0.9226	0.9022
Average					0.8604	0.8829	0.9455	0.9130

Berdasarkan hasil perhitungan metrik evaluasi pada setiap fold, diperoleh nilai akurasi, *precision*, *recall*, dan *f1-score* yang berbeda-

beda. Dari kelima *fold* tersebut kemudian dirata-ratakan seperti pada Tabel 4.9 untuk mendapatkan nilai keseluruhan (*overall*) akurasi, precision, *recall*, dan *f1-score* sebagai representasi kinerja model secara umum. Pada Tabel 4.9 menunjukkan hasil metrik evaluasi secara keseluruhan dengan nilai akurasi sebesar 0.8604, *precision* 0.8829, *recall* 0.9455, dan *f1-score* 0.9130.

4.3.2 Model TextCNN Sedang

Untuk mengevaluasi performa model TextCNN pada setiap fold, digunakan *confusion matrix* yang berisi jumlah prediksi benar maupun salah dari kelas positif dan negatif. Matriks ini merepresentasikan nilai *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), dan *False Negative* (FN).

Tabel 4.10 Hasil Evaluasi Model TextCNN Sedang

Fold	**Confusion Matrix**				**Metrik Evaluasi**			
	TP	TN	FP	FN	Acc	Pre	Rec	F1
0	724	158	68	50	0.882	0.9141	0.9354	0.9246
1	736	127	98	39	0.863	0.8825	0.9497	0.9149
2	718	157	68	57	0.875	0.9135	0.9265	0.9199
3	730	140	85	45	0.870	0.8957	0.9419	0.9182
4	729	132	93	46	0.861	0.8869	0.9406	0.9130
Average					0.8768	0.8985	0.9388	0.9181

Berdasarkan hasil perhitungan metrik evaluasi pada setiap fold, diperoleh nilai akurasi, *precision*, *recall*, dan *f1-score* yang berbeda-beda. Dari kelima fold tersebut kemudian dirata-ratakan seperti pada Tabel 4.10 untuk mendapatkan nilai keseluruhan (*overall*) akurasi, *precision*, *recall*, dan *f1-score* sebagai representasi kinerja model secara umum. Pada Tabel 4.10 menunjukkan hasil metrik evaluasi secara

keseluruhan dengan nilai akurasi sebesar 0.8768, *precision* 0.8985, *recall* 0.9388, dan *f1-score* 0.9181.

4.3.3 Model TextCNN Berat

Untuk mengevaluasi performa model TextCNN pada setiap fold, digunakan *confusion matrix* yang berisi jumlah prediksi benar maupun salah dari kelas positif dan negatif. Matriks ini merepresentasikan nilai *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), dan *False Negative* (FN).

Tabel 4.11 Hasil Evaluasi Model TextCNN Berat

Fold	**Confusion Matrix**				**Metrik Evaluasi**			
	TP	TN	FP	FN	Acc	Pre	Rec	F1
0	736	149	77	38	0.885	0.9053	0.9509	0.9275
1	730	146	79	45	0.876	0.9023	0.9419	0.9217
2	730	159	66	45	0.889	0.9171	0.9419	0.9293
3	696	178	47	97	0.874	0.9367	0.8981	0.9170
4	706	152	73	69	0.858	0.9063	0.9110	0.9086
Average					0.8764	0.9135	0.9288	0.9205

Berdasarkan hasil perhitungan metrik evaluasi pada setiap fold, diperoleh nilai akurasi, *precision*, *recall*, dan *f1-score* yang berbeda-beda. Dari kelima fold tersebut kemudian dirata-ratakan seperti pada Tabel 4.11 untuk mendapatkan nilai keseluruhan (*overall*) akurasi, *precision*, *recall*, dan *f1-score* sebagai representasi kinerja model secara umum. Pada Tabel 4.11 menunjukkan hasil metrik evaluasi secara keseluruhan dengan nilai akurasi sebesar 0.8768, *precision* 0.8985, *recall* 0.9388, dan *f1-score* 0.9181.

4.3.4 Model PAT-CNN

Untuk mengevaluasi performa model PAT-CNN pada setiap fold, digunakan *confusion matrix* yang berisi jumlah prediksi benar maupun salah dari kelas positif dan negatif. Matriks ini merepresentasikan nilai *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), dan *False Negative* (FN).

Tabel 4.12 Hasil Evaluasi Model PAT-CNN

Fold	**Confusion Matrix**				**Metrik Evaluasi**			
	TP	TN	FP	FN	Acc	Pre	Rec	F1
0	721	155	77	53	0.876	0.9104	0.9315	0.9208
1	723	155	70	52	0.878	0.9117	0.9329	0.9222
2	697	183	42	78	0.880	0.9432	0.8994	0.9207
3	698	174	51	77	0.872	0.9319	0.9006	0.9160
4	708	158	67	67	0.866	0.9135	0.9135	0.9135
Average					0.8744	0.9221	0.9156	0.9186

Berdasarkan hasil perhitungan metrik evaluasi pada setiap fold, diperoleh nilai akurasi, *precision*, *recall*, dan *f1-score* yang berbeda-beda. Dari kelima fold tersebut kemudian dirata-ratakan seperti pada Tabel 4.12 untuk mendapatkan nilai keseluruhan (*overall*) akurasi, *precision*, *recall*, dan *f1-score* sebagai representasi kinerja model secara umum. Pada Tabel 4.12 menunjukkan hasil metrik evaluasi secara keseluruhan dengan nilai akurasi sebesar 0.8744, *precision* 0.9221, *recall* 0.9156, dan *f1-score* 0.9186.

4.3.5 Perbandingan Evaluasi Model

Perbandingan evaluasi antar *fold* pada keempat model menunjukkan pola konsistensi yang menarik dalam konteks *cross-validation*. Variasi performa yang terjadi antar *fold* dapat dijelaskan

melalui perbedaan distribusi karakteristik ulasan dalam subset data yang berbeda. Ketika akurasi mengalami penurunan pada *fold* tertentu, hal ini menunjukkan bahwa *validation set* mengandung sampel ulasan dengan kompleksitas linguistik lebih tinggi atau pola sentimen yang lebih ambigu. Sebaliknya, peningkatan akurasi mengindikasikan *validation set* memiliki karakteristik yang lebih representatif dengan *training set*.

Secara khusus, fold 2 menunjukkan performa terbaik pada hampir semua model, dengan TextCNN Ringan mencapai akurasi 0.867, TextCNN Sedang 0.875, TextCNN Berat 0.889, dan PAT-CNN 0.880. Fenomena ini dapat dijelaskan melalui karakteristik distribusi data dalam dataset ulasan Shopee. Pembagian data menggunakan *stratified k-fold* dengan *random state* 2025 menghasilkan konfigurasi pada *fold* 2 yang memiliki keseimbangan optimal antara representasi *sentiment classes* dan variasi *linguistic patterns*. *Training set* pada iterasi ini mengandung sampel yang lebih representatif dengan proporsi seimbang antara ulasan dengan sentimen eksplisit dan implisit, sehingga model dapat melakukan prediksi dengan *confidence* lebih tinggi. Dataset dengan dominasi ulasan positif (69.88% rating 5 bintang) menciptakan tantangan *class imbalance* yang mempengaruhi kemampuan model, terutama pada *fold* dengan distribusi yang kurang seimbang.

Model TextCNN Berat dan PAT-CNN menunjukkan performa yang tidak jauh berbeda dengan model sederhana ketika dikaitkan dengan karakteristik dataset. TextCNN Berat mencapai akurasi 0.8764 dengan *f1-score* 0.9205, sementara PAT-CNN menghasilkan akurasi 0.8744 dengan *precision* tertinggi 0.9221. *Performance gap* yang kecil ini dapat dijelaskan oleh beberapa faktor seperti ukuran dataset 5000

ulasan mungkin tidak cukup besar untuk memanfaatkan penuh kapasitas model yang sangat kompleks, karakteristik ulasan Shopee yang cenderung *straightforward* dalam ekspresi sentimen tidak memerlukan *architectural sophistication* tinggi, *preprocessing* yang komprehensif telah menyederhanakan *complexity* data input, dan regulasi melalui *dropout* membatasi kemampuan model kompleks.

Stabilitas performa yang terlihat menunjukkan bahwa penggunaan 5 *fold cross-validation* sudah mencukupi untuk evaluasi yang *reliable*, tanpa perlu memperluas hingga 10 *fold validation*. Rentang akurasi antar *fold* yang relatif minim (0.8604–0.8768) menunjukkan model telah mencapai tingkat performa yang stabil, dan konsistensi performa model antar *fold* menunjukkan bahwa penggunaan 5 *fold* sudah cukup untuk menghasilkan perbandingan yang andal. Penambahan jumlah *fold* hanya akan meningkatkan beban komputasi tanpa memberikan peningkatan yang signifikan dalam evaluasinya.

4.3.6 *Ablation Study*

Ablation study dilakukan untuk mengevaluasi pengaruh hyperparameter seperti *optimizer*, *learning rate*, dan *dropout* terhadap performa model dengan tujuan mengidentifikasi konfigurasi paling optimal serta menilai sensitivitas model terhadap variasi parameter. Model TextCNN Sedang dengan 10.08M parameter dan *embedding* 300D dipilih karena memberikan keseimbangan antara kompleksitas arsitektur, *baseline performance* yang cukup tinggi yaitu akurasi 0.8746, dan kompleksitas komputasi yang masih efisien untuk *multiple experimental runs*. Sebaliknya, Model TextCNN Berat meskipun memiliki kapasitas lebih besar dengan 17.20M parameter menunjukkan efisiensi parameter yang rendah. Model Ringan meskipun

komputasinya lebih ringan tetapi menghasilkan *baseline performance* yang rendah yaitu akurasi 0.8606, sementara Model PAT-CNN dengan arsitektur kompleks dinilai kurang tepat untuk isolasi pengaruh *hyperparameter* secara spesifik. Hasil evaluasi variasi *hyperparameter* dapat dilihat pada Tabel 4.13 Hasil Ablation Study.

Tabel 4.13 Hasil Ablation Study

Optimizers	**Learning Rate**	**Dropout**	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Muon	0.0001	0.2	0.8820	0.9141	0.9354	0.9246
	0.001		0.8770	0.9094	0.9341	0.9216
	0.0005		0.8820	**0.9271**	0.9199	0.9235
	0.005		**0.8870**	0.9147	**0.9419**	**0.9281**
Adam	0.0001	0.0	0.8830	0.9041	**0.9496**	0.9263
		0.1	0.8840	0.9186	0.9328	0.9256
		0.3	0.8870	0.9168	0.9393	0.9279
		0.4	**0.8890**	**0.9170**	0.9419	0.9293
		0.5	0.8890	0.9139	0.9457	**0.9295**
Muon	0.0001	0.2	0.8450	**0.8923**	0.9096	0.9008
	0.001		**0.8780**	0.8909	0.9599	**0.9241**
	0.0005		0.8710	0.8826	**0.9612**	0.9202
	0.005		0.8450	0.8578	0.9587	0.9054
Muon	0.0001	0.0	**0.8490**	0.8635	**0.9561**	**0.9074**
		0.1	0.8400	0.8717	0.9302	0.9000
		0.3	0.8440	**0.8862**	0.9160	0.9009
		0.4	0.8470	0.8848	0.9225	0.9032
		0.5	0.8410	0.8588	0.9509	0.9025

Hasil eksperimen menunjukkan bahwa *optimizer* memiliki pengaruh paling signifikan terhadap *performance* model. Adam secara konsisten mengungguli Muon dengan margin yang cukup besar, mencapai akurasi 0.8820 dibandingkan 0.8450 untuk Muon dengan selisih 3.7% dan *f1-score* 0.9246 serta 0.9008. *Learning rate* menunjukkan pola performa yang stabil, dimana 0.0001 memberikan *performance* terbaik yaitu akurasi 0.8820, *f1-score* 0.9246, sementara nilai 0.001, 0.0005, dan 0.005 memberikan hasil identik, menunjukkan

kestabilan model terhadap variasi *learning rate* dalam rentang tersebut. *Dropout rate* menunjukkan bahwa model benefit dari *regularization* agresif, dengan *dropout* 0.0 menghasilkan *recall* tinggi yaitu 0.9496 namun *precision moderate* yaitu 0.9041 yang mengindikasikan *overfitting*, sedangkan *dropout* optimal berada di rentang 0.4-0.5 dengan akurasi tertinggi 0.8890 dan *f1-score* 0.9293.

Ranking sensitivitas parameter menunjukkan *optimizer choice* memiliki *impact* terbesar (3.7% *difference*), diikuti *dropout rate* (0.6% variasi), dan *learning rate* paling tidak sensitif (0.5% *difference*). *Interaction effects* mengkonfirmasi bahwa *learning rate* memiliki *plateau effect* yang lebar memberikan fleksibilitas *tuning*, dropout menunjukkan *sweet spot* di 0.4-0.5 untuk mencegah *overfitting*, dan Adam menunjukkan stabilitas lebih baik *across configurations* dibandingkan Muon. Berdasarkan hasil *ablation study*, konfigurasi optimal adalah *learning rate* 0.0001, *optimizer* Adam, dan *dropout rate* 0.4, yang diproyeksikan memberikan akurasi sekitar 0.8890 dan *f1-score* 0.9293.

4.3.7 Kompleksitas Komputasi

Evaluasi kompleksitas komputasi merupakan aspek kritis dalam pengembangan model *deep learning*, terutama untuk aplikasi praktis yang memerlukan pertimbangan efisiensi *resource* dan *deployment constraints*. Dalam konteks *text classification* menggunakan TextCNN, kompleksitas komputasi tidak hanya ditentukan oleh jumlah parameter model, tetapi juga dipengaruhi oleh kompleksitas arsitektur, jenis operasi yang dilakukan, dan struktur layer yang diimplementasikan. Analisis ini mengukur tiga metrik utama: jumlah parameter total yang menentukan *memory requirement*, *Multiply-Accumulate Operations* (MACs) yang mengindikasikan *computational complexity* selama

inference, dan ukuran model dalam megabyte yang mempengaruhi *storage requirement* dan *loading time*. Hasil perbandingan kompleksitas komputasi dapat dilihat pada Tabel 4.14.

Tabel 4.14 Perbandingan Hasil Kompleksitas Komputasi

Model	# Params	MACs (M)	Size (MB)
TextCNN Ringan	3.361.606	1330	13.45
TextCNN Sedang	10.080.206	3980	40.32
TextCNN Berat	17.201.922	6790	68.81
PAT-CNN	8.269.501	7670	33.08

Pada Tabel 4.14 terlihat perbandingan antara kompleksitas model dan efisiensi komputasi. Model TextCNN Ringan dengan 3.361.606 parameter menghasilkan kompleksitas komputasi terendah dengan 1330 MACs dan ukuran model 13.45 MB, menjadikannya pilihan optimal untuk *deployment* dengan keterbatasan *resource*. Model TextCNN Sedang dengan 10.080.206 parameter menunjukkan peningkatan kompleksitas komputasi yang proporsional dengan 3980 MACs dan 40.32 MB, memberikan keseimbangan antara *performance* dan efisiensi. Model TextCNN Berat mencapai 17.201.922 parameter dengan 6790 MACs dan 68.81 MB, namun menunjukkan efisiensi yang buruk mengingat *performance* yang tidak sebanding dengan *resource* yang dibutuhkan. Model PAT-CNN, meskipun memiliki parameter count yang moderat yaitu 8.269.501, menghasilkan kompleksitas komputasi tertinggi dengan 7670 MACs dan 33.08 MB, yang dapat dijelaskan oleh kompleksitas arsitektur dengan *attention mechanisms* dan *squeeze-excitation blocks* yang memerlukan operasi tambahan. Hasil ini menunjukkan bahwa *parameter coun*t bukan satu-satunya indikator kompleksitas komputasi, dimana *architectural complexity*

model PAT-CNN menghasilkan MACs tertinggi meskipun parameter countnya lebih rendah dari Model Berat, menunjukkan pentingnya pertimbangan *architectural efficiency*.

4.4 Analisis Hasil Penelitian

Tahap ini merupakan tahap terakhir pada penelitian yang akan membahas hasil penelitian yang telah dilakukan terhadap performa keempat model yang digunakan untuk klasifikasi sentimen pada ulasan aplikasi Shopee. Penelitian ini mengembangkan empat variasi model dengan tujuan mengeksplorasi keseimbangan antara kompleksitas arsitektur, efisiensi komputasi, dan performa klasifikasi dalam konteks analisis sentimen pada domain *e-commerce* Indonesia. Pengembangan beberapa model dengan kapasitas yang berbeda dilakukan untuk menganalisis secara sistematis konfigurasi model yang paling optimal untuk dataset yang digunakan. Hasil seluruh evaluasi yang telah dilakukan dapat dilihat pada Tabel 4.15.

Tabel 4.15 Hasil Pengujian Model

Model	**Akurasi**	***Precision***	***Recall***	***F1-Score***
TextCNN Ringan	0.8604	0.8829	0.9455	0.9130
TextCNN Sedang	0.8768	0.8985	0.9388	0.9181
TextCNN Berat	0.8764	0.9135	0.9288	0.9205
PAT-CNN	0.8744	0.9221	0.9156	0.9186

Berdasarkan hasil pengujian pada Tabel 4.15, terlihat bahwa setiap varian model TextCNN memberikan performa yang berbeda pada masing-masing metrik evaluasi. Model TextCNN Ringan memperoleh nilai akurasi sebesar 0.8604 dengan *precision* 0.8829, *recall* tertinggi sebesar 0.9455, serta *f1-score* sebesar 0.9130. Hal ini menunjukkan bahwa TextCNN Ringan memiliki kemampuan yang baik dalam

mendeteksi data sesuai label sebenarnya, meskipun nilai akurasi dan *precision* lebih rendah dibandingkan model lainnya. Sementara itu, TextCNN Sedang menghasilkan akurasi tertinggi yaitu 0.8768 dengan *precision* 0.8985, *recall* 0.9388, dan *f1-score* 0.9181, yang menggambarkan performa yang relatif seimbang pada seluruh metrik. Adapun TextCNN Berat menghasilkan akurasi 0.8764 dengan *precision* tertinggi sebesar 0.9135, *recall* 0.9288, serta *f1-score* tertinggi sebesar 0.9205. Model PAT-CNN menghasilkan akurasi 0.8744 dengan *precision* 0.9221, *recall* 0.9156, dan *f1-score* 0.9186, menunjukkan performa yang kompetitif dengan fokus pada *precision* yang tinggi.

Meskipun terdapat variasi performa antar model, perbedaan yang dihasilkan relatif tidak signifikan dengan rentang akurasi hanya 0.0164 dan *f1-score* dalam rentang 0.0075. Hal ini dapat dijelaskan oleh beberapa faktor yang berkaitan dengan dataset yang digunakan. Klasifikasi sentimen pada ulasan produk Shopee merupakan domain yang relatif *straightforward* dimana pola bahasa sentimen cenderung eksplisit dan mudah diidentifikasi, sehingga tidak memerlukan kompleksitas arsitektur yang tinggi untuk mencapai performa optimal. Dataset dengan 5000 ulasan sudah mencukupi untuk model sederhana dalam mencapai batas performa maksimal, dimana penambahan parameter atau kompleksitas arsitektur tidak memberikan peningkatan yang signifikan karena telah mencapai puncak performa. Model yang lebih kompleks seperti TextCNN Berat dengan 17.2M parameter atau PAT-CNN dengan *attention mechanisms* berpotensi mengalami *overfitting* pada dataset berukuran sedang ini, sementara *regularization* yang diterapkan membatasi kemampuan model untuk memanfaatkan kapasitas penuh mereka. Karakteristik spesifik ulasan produk yang menggunakan bahasa informal dan *pattern repetitive* lebih cocok

ditangani oleh *convolutional layers* sederhana yang mampu menangkap n-gram *patterns* relevan tanpa memerlukan *enhanced architectural components* seperti *attention* atau *squeeze-excitation blocks*, sehingga *architectural innovations* tidak memberikan *discriminative power* tambahan yang signifikan pada domain ini.

Analisis perbandingan metrik performa memberikan temuan utama untuk pemilihan model dalam implementasi nyata. TextCNN Ringan dengan *recall* tertinggi 0.9455 dan *computational efficiency* terbaik (1330 MACs, 13.45 MB) merupakan pilihan optimal untuk *resource-constrained environments* dimana mendeteksi seluruh sentimen positif menjadi prioritas utama. TextCNN Sedang dengan *balanced performance* yang menghasilkan akurasi 0.8768 dan *f1-score* 0.9181, serta kebutuhan komputasi yang moderat (3980 MACs, 40.32 MB), merepresentasikan titik optimal antara performa dan efisiensi sehingga menjadi pilihan paling serbaguna untuk aplikasi analisis sentimen secara umum. TextCNN Berat dengan *f1-score* tertinggi 0.9205 cocok untuk aplikasi dimana akurasi prediksi menjadi faktor utama dan sumber daya komputasi bukanlah kendala. Sementara itu, PAT-CNN dengan *precision* tertinggi 0.9221, namun biaya komputasi yang signifikan (7670 MACs) lebih tepat digunakan pada kasus khusus yang membutuhkan tingkat keyakinan tinggi dalam prediksi positif.

Kesimpulannya, penelitian ini berhasil mendemonstrasikan bahwa arsitektur TextCNN dapat diterapkan secara efektif untuk klasifikasi sentimen ulasan *e-commerce* dengan performa yang memuaskan pada berbagai konfigurasi model. Konsistensi performa antar model mengindikasikan bahwa untuk tugas analisis sentimen pada domain *e-commerce* Indonesia dengan ukuran dataset yang cukup representatif, arsitektur yang lebih sederhana sudah memberikan

performa yang mendekati optimal, dan penambahan kompleksitas arsitektur memberikan peningkatan yang semakin berkurang. Penelitian ini menjawab rumusan masalah dengan menunjukkan secara konklusif bahwa TextCNN mencapai klasifikasi sentimen yang reliabel dengan akurasi berkisar 86-88% dan *f1-score* 91-92%, yang divalidasi melalui 5 *fold cross-validation* untuk memastikan kemampuan generalisasi. Tahap *preprocessing* yang komprehensif meliputi normalisasi, augmentasi, dan penghapusan *stopwords* terbukti krusial dalam mencapai performa optimal,sementara strategi validasi *k-fold* memberikan kerangka evaluasi yang andal untuk menilai konsistensi dan reliabilitas model dalam skenario implementasi nyata

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil penelitian yang telah dilakukan, arsitektur TextCNN berhasil diterapkan secara efektif untuk analisis sentimen ulasan pengguna aplikasi Shopee di Google Play Store dengan kinerja yang memuaskan. Implementasi empat variasi model yaitu TextCNN Ringan, Sedang, Berat, dan PAT-CNN dengan evaluasi menggunakan 5 *fold cross validation* menunjukkan bahwa TextCNN Sedang mencapai performa terbaik dengan akurasi 87.68%, presisi 89.85%, *recall* 93.88%, dan *f1-score* 91.81%. Model TextCNN Berat menghasilkan *f1-score* tertinggi 92.05% dengan presisi 91.35%, sementara PAT-CNN menunjukkan presisi terbaik 92.21% meski dengan kompleksitas komputasi yang lebih tinggi. Evaluasi *stratified k-fold cross validation* terbukti efektif dalam mengukur konsistensi performa model, dimana seluruh model menunjukkan stabilitas yang baik dengan variasi performa antar *fold* yang relatif kecil dengan rentang akurasi 1.64%. *Preprocessing data* yang komprehensif meliputi normalisasi, augmentasi teks, dan penghapusan *stopwords* berkontribusi signifikan terhadap pencapaian performa optimal, sehingga penelitian ini berhasil menjawab rumusan masalah dengan mendemonstrasikan bahwa TextCNN dapat diterapkan dan diukur kinerjanya secara akurat untuk klasifikasi sentimen ulasan *e-commerce* menggunakan metrik evaluasi standar dan validasi silang.

5.2 Saran

Berdasarkan hasil penelitian, beberapa aspek dapat dikembangkan pada penelitian selanjutnya adalah.

1. Ekspansi dataset dengan jumlah lebih besar dan rentang waktu lebih panjang untuk menguji skalabilitas model.
2. Pengembangan klasifikasi multi-kelas atau berbasis aspek untuk analisis yang lebih detail terhadap fitur spesifik seperti pengiriman dan layanan pelanggan.
3. Eksplorasi *transfer learning* menggunakan *pre-trained models* seperti IndoBERT untuk meningkatkan representasi kontekstual.
4. Penerapan teknik *advanced* untuk menangani *class imbalance* seperti SMOTE atau *focal loss* untuk meningkatkan deteksi sentimen negatif.
5. implementasi sistem *monitoring real-time* dengan *deployment* model dalam *production environment* untuk aplikasi praktis dalam *customer feedback analysis* dan *business initial value.*

DAFTAR PUSTAKA

[1] T. Hendro Pudjiantoro, F. Rakhmat Umbara, B. Trihatmoko, and U. Jenderal Achmad Yani Jl Terusan Sudirman, "Analisis Sentimen Terhadap E-commerce Pada Media Sosial Twitter Menggunakan Metode Naïve bayes," *Seminar Nasional Informatika dan Aplikasinya (SNIA)*, p. 2021, Aug. 2021.

[2] A. Ahdiat, "5 E-Commerce dengan Pengunjung Terbanyak Sepanjang 2023," Databoks.

[3] A. Puspita, D. Finanta, and C. Rayani Ibrahim Sinulingga, "PERAN TEKNOLOGI DIGITAL DALAM MENINGKATKAN EFESIENSI TRANSAKSI E-COMMERCE SHOPEE," *Jurnal Sains Student Research*, vol. 3, no. 1, 2025, doi: 10.61722/jssr.v3i1.3240.

[4] Author F, "Mengapa Pengguna Android Lebih Banyak Dibandingkan iOS di Indonesia," JalanTeknologi.

[5] C. Prakoso and A. Hermawan, "Perbandingan Model Machine Learning dalam Analisis Sentimen Ulasan Pengunjung Keraton Yogyakarta pada Google Maps," *Kajian Ilmiah Informatika dan Komputer*, vol. 4, no. 3, 2023.

[6] D. Irawan *et al.*, "KLASIFIKASI SENTIMEN PELANGGAN RESTORAN KOKI SUNDA MENGGUNAKAN ALGORITMA NAIVE BAYES," *Jurnal Manajemen Informatika & Sistem Informasi (MISI)*, vol. 8, no. 2, 2025, doi: 10.36595/misi.v5i2.

[7] A. Rahul, S. Giriprasad, G. Neelamadhav, and M. Senthil, "Fault in your stars: An Analysis of Android App Reviews," in

Proceedings of the 2018 ACM India Joint International Conference on Data Science and Management of Data (CoDS-COMAD), ACM, 2018.

[8] H. Nguyen, A. Veluchamy, M. Diop, and R. Iqbal, "Comparative Study of Sentiment Analysis with Product Reviews Using Machine Learning and Lexicon-Based Approaches," *SMU Data Science Review*, vol. 1, no. 4, [Online]. Available: https://scholar.smu.edu/datasciencereviewAvailableat:https://scholar.smu.edu/datasciencereview/vol1/iss4/7http://digitalrepository.smu.edu.

[9] O. E. Putri *et al.*, "ANALISIS SENTIMEN BERBASIS ASPEK DAN DETEKSI EMOSI PADA COFFEE SHOP PALANGKA RAYA MENGGUNAKAN DEEP LEARNING," *JOINTECOMS (Journal of Information Technology and Computer Science) p-ISSN: 2798-284X*, vol. 4, no. 3, pp. 2798–3862, 2024.

[10] V. Fitriyana, Lutfi Hakim, Dian Candra Rini Novitasari, and Ahmad Hanif Asyhar, "Analisis Sentimen Ulasan Aplikasi Jamsostek Mobile Menggunakan Metode Support Vector Machine," *Jurnal Buana Informatika*, vol. 14, no. 01, pp. 40–49, Apr. 2023, doi: 10.24002/jbi.v14i01.6909.

[11] S. Christina, U. Palangka Raya, J. Yos Sudarso Kel Palangka Kec Jekan Raya, P. Raya, and K. Tengah, "Sarcasm in Sentiment Analysis of Indonesian Text: A Literature Review," *Jurnal Teknologi Informasi*, vol. 13, no. 2, Aug. 2019.

[12] H.-T. Duong and T.-A. Nguyen-Thi, "A review: preprocessing techniques and data augmentation for sentiment analysis,"

Comput Soc Netw, vol. 8, no. 1, p. 1, 2021, doi: 10.1186/s40649-020-00080-x.

[13] A. Maskhuri and T. Hardiani, "Analisis Sentimen Pengguna pada Aplikasi Tokopedia Menggunakan Algoritma Convolutional Neural Network," *Technology and Science (BITS)*, vol. 6, no. 4, 2025, doi: 10.47065/bits.v6i4.6923.

[14] N. C. Dang, M. N. Moreno-García, and F. De la Prieta, "Sentiment analysis based on deep learning: A comparative study," *Electronics (Switzerland)*, vol. 9, no. 3, Mar. 2020, doi: 10.3390/electronics9030483.

[15] R. Wesley and R. Gunawan, "LITERATUR REVIEW: METODE DEEP LEARNING UNTUK ANALISIS TEKS," *Jurnal Mahasiswa Teknik Informatika*, vol. 8, no. 5, pp. 11020–11023, Oct. 2024.

[16] Y. Kong *et al.*, "Text classification in fair competition law violations using deep learning," *Front Appl Math Stat*, vol. 9, Sep. 2023, doi: 10.3389/fams.2023.1177081.

[17] C. Zhang, R. Guo, X. Ma, X. Kuai, and B. He, "W-TextCNN: A TextCNN model with weighted word embeddings for Chinese address pattern classification," *Comput Environ Urban Syst*, vol. 95, 2022, doi: 10.1016/j.compenvurbsys.2022.101819.

[18] Y. E. Mahendra, R. Ilyas, and F. Kasyidi, "Klasifikasi Kalimat Ilmiah Menggunakan 1D Convolutional Neural Networks," *Prosiding The 11th Industrial Research Workshop and National Seminar*, pp. 503–509, Aug. 2020.

[19] A. U. Rehman, A. K. Malik, B. Raza, and W. Ali, “A Hybrid CNN-LSTM Model for Improving Accuracy of Movie Reviews Sentiment Analysis,” *Multimed Tools Appl*, vol. 78, no. 18, pp. 26597–26613, Sep. 2019, doi: 10.1007/s11042-019-07788-7.

[20] Y. N. FUADAH, I. D. UBAIDULLAH, N. IBRAHIM, F. F. TALININGSING, N. K. SY, and M. A. PRAMUDITHO, “Optimasi Convolutional Neural Network dan K-Fold Cross Validation pada Sistem Klasifikasi Glaukoma,” *ELKOMIKA: Jurnal Teknik Energi Elektrik, Teknik Telekomunikasi, & Teknik Elektronika*, vol. 10, no. 3, p. 728, Jul. 2022, doi: 10.26760/elkomika.v10i3.728.

[21] D. Normawati and S. A. Prayogi, “Implementasi Naïve Bayes Classifier Dan Confusion Matrix Pada Analisis Sentimen Berbasis Teks Pada Twitter,” *Jurnal Sains Komputer & Informatika (J-SAKTI*, vol. 5, no. 2, pp. 697–711, Sep. 2021.

[22] E. Y. Hidayat and D. Handayani, “Penerapan 1D-CNN untuk Analisis Sentimen Ulasan Produk Kosmetik Berdasar Female Daily Review,” *Jurnal Nasional Teknologi dan Sistem Informasi*, vol. 8, no. 3, pp. 153–163, Jan. 2023, doi: 10.25077/teknosi.v8i3.2022.153-163.

[23] R. Naquitasia, D. Hatta Fudholi, and L. Iswari, “ANALISIS SENTIMEN BERBASIS ASPEK PADA WISATA HALAL DENGAN METODE DEEP LEARNING,” *JURNAL TEKNOINFO*, vol. 16, no. 2, pp. 156–164, 2022, [Online]. Available: https://ejurnal.teknokrat.ac.id/index.php/teknoinfo/index

[24] I. Azizah, I. Cholissodin, and N. Yudistira, “Analisis Sentimen Ulasan Pengguna Aplikasi Shopee di Google Play menggunakan Metode Word Embedding dan Long Short Term Memory (LSTM),” *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, vol. 7, no. 5, pp. 2453–2459, 2023, [Online]. Available: http://j-ptiik.ub.ac.id

[25] D. Purnamasari *et al.*, *Pengantar Metode Analisis Sentimen*. Penerbit Gunadarma, 2023.

[26] B. Gunawan, H. S. Pratiwi, and E. E. Pratama, “Sistem Analisis Sentimen pada Ulasan Produk Menggunakan Metode Naive Bayes,” *Jurnal Edukasi dan Penelitian Informatika (JEPIN)*, vol. 4, no. 2, pp. 17–29, Dec. 2018, [Online]. Available: www.femaledaily.com

[27] E. M. Asih, “Analisis pada Shopee sebagai E-Commerce Terpopuler di Indonesia,” *Jurnal Ekonomi Bisnis Antartika*, vol. 2, pp. 2024–73, Jun. 2024.

[28] F. I. Wibowo and A. Febriandirza, “Analisis Sentimen Ulasan Pengguna Game Pubg Di Google Play Store Menggunakan Algoritma Naïve Bayes,” *Jurnal Sistem Komputer dan Informatika (JSON)*, vol. 5, no. 3, p. 590, Mar. 2024, doi: 10.30865/json.v5i3.7264.

[29] T. Gori, A. Sunyoto, and H. Al Fatta, “Preprocessing Data dan Klasifikasi untuk Prediksi Kinerja Akademik Siswa,” *Jurnal Teknologi Informasi dan Ilmu Komputer*, vol. 11, no. 1, pp. 215–224, Feb. 2024, doi: 10.25126/jtiik.20241118074.

[30] B. Farasqa Nauval Akbar, D. Arifianto, A. Maryam Zakiyyah, and A. Milu Susetyo, “Analisis Sentimen Terhadap Anies Baswedan Menggunakan Metode Support Vector Machine Studi Kasus Media Sosial Twitter Sentiment Analysis of Anies Baswedan Using the Support Vector Machine Method Case Study of Twitter Social Media,” 2021.

[31] M. A. Ganaie, M. Hu, A. K. Malik, M. Tanveer, and P. N. Suganthan, “Ensemble deep learning: A review,” 2022. doi: 10.1016/j.engappai.2022.105151.

[32] W. Xing and D. Du, “Dropout Prediction in MOOCs: Using Deep Learning for Personalized Intervention,” *Journal of Educational Computing Research*, vol. 57, no. 3, pp. 547–570, Jun. 2019, doi: 10.1177/0735633118757015.

[33] S. Gholizadeh, “Top Popular Python Libraries in Research,” 2022. [Online]. Available: www.opastonline.com

[34] M. Wang and F. Hu, “The application of nltk library for python natural language processing in corpus research,” *Theory and Practice in Language Studies*, vol. 11, no. 9, pp. 1041–1049, Sep. 2021, doi: 10.17507/tpls.1109.09.

[35] Q. Wang and Q. Qian, “Malicious code classification based on opcode sequences and textCNN network,” *Journal of Information Security and Applications*, vol. 67, Jun. 2022, doi: 10.1016/j.jisa.2022.103151.

[36] A. Rawat, M. A. Wani, M. ElAffendi, A. S. Imran, Z. Kastrati, and S. M. Daudpota, “Drug Adverse Event Detection Using Text-Based Convolutional Neural Networks (TextCNN) Technique,”

Electronics (Switzerland), vol. 11, no. 20, Oct. 2022, doi: 10.3390/electronics11203336.

[37] B. Guo, C. Zhang, J. Liu, and X. Ma, "Improving text classification with weighted word embeddings via a multi-channel TextCNN model," *Neurocomputing*, vol. 363, pp. 366–374, Oct. 2019, doi: 10.1016/j.neucom.2019.07.052.

[38] O. Hrinchuk, V. Khrulkov, L. Mirvakhabova, E. Orlova, and I. Oseledets, "Tensorized Embedding Layers," *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4847–4860, Nov. 2020, doi: 10.18653/v1/2020.findings-emnlp.436.

[39] M. Susanty and S. Sukardi, "Perbandingan Pre-trained Word Embedding dan Embedding Layer untuk Named-Entity Recognition Bahasa Indonesia," *PETIR: Jurnal Pengkajian dan Penerapan Teknik Informatika*, vol. 14, no. 2, pp. 247–257, Sep. 2021, doi: 10.33322/petir.v14i2.1164.

[40] L. Lu, Y. Zheng, G. Carneiro, and L. Yang, *Deep Learning and Convolutional Neural Networks for Medical Image Computing: Precision Medicine, High Performance and Large-Scale Datasets*. 2017. doi: 10.1007/978-3-319-42999-1.

[41] B. Zhang, Q. Zhao, W. Feng, and S. Lyu, "AlphaMEX: A smarter global pooling method for convolutional neural networks," *Neurocomputing*, vol. 321, pp. 36–48, 2018, doi: https://doi.org/10.1016/j.neucom.2018.07.079.

[42] K. R. Wardani and L. Leonardi, "Klasifikasi Penyakit pada Daun Anggur menggunakan Metode Convolutional Neural Network,"

Jurnal Tekno Insentif, vol. 17, no. 2, pp. 112–126, Oct. 2023, doi: 10.36787/jti.v17i2.1130.

[43] R. M. Aldani, A. Bhirawa, and U. Pradema Sanjaya, "OPTIMASI KLASIFIKASI DATA TIDAK SEIMBANG PADA DATASET MEDIS PADA KASUS PENYAKIT GINJAL KRONIS DENGAN TEKNIK SMOTE," *JURNAL INOVTEK POLBENG - SERI INFORMATIKA*, vol. 10, no. 1, p. 2025, Mar. 2025.

[44] G. P. Insany, I. L. Kharisma, and R. Rosmawati, "Penerapan Algoritma Random Forest untuk Menganalisis Ulasan Aplikasi Spotify pada Google Play," *Edumatic: Jurnal Pendidikan Informatika*, vol. 8, no. 2, pp. 369–378, Dec. 2024, doi: 10.29408/edumatic.v8i2.26394.

[45] X. Zhang, J. Zhao, and Y. LeCun, "Character-level Convolutional Networks for Text Classification," *Advances in Neural 96 Information Processing Systems 28 (NIPS 2015)*, pp. 649–657, Apr. 2016, [Online]. Available: http://arxiv.org/abs/1502.01710

[46]. Usha Ruby Dr.A, "Binary cross entropy with deep learning technique for Image classification," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 9, no. 4, pp. 5393–5397, Aug. 2020, doi: 10.30534/ijatcse/2020/175942020.

[47] D. Arifuddin, K. Kusrini, and K. Kusnawi, "Perbandingan Performansi Algoritma Multiple Linear Regression dan Multi Layer Perceptron Neural Network dalam Memprediksi Penjualan Obat," *MALCOM: Indonesian Journal of Machine Learning and*

Computer Science, vol. 5, no. 2, pp. 722–737, Apr. 2025, doi: 10.57152/malcom.v5i2.1952.

[48] N. Sarasuartha Mahajaya, P. Desiana, W. Ayu, and R. R. Huizen, "Pengaruh Optimizer Adam, AdamW, SGD, dan LAMB terhadap Model Vision Transformer pada Klasifikasi Penyakit Paru-paru," *SPINTER 2024*, vol. 1, no. 2, pp. 818–823, Apr. 2024, [Online]. Available: https://www.kaggle.com/datasets/tawsifurrahman/covid19-radiography-database,