Jurnal Teknik Informatika (JUTIF)
Vol. 5, No. 5, October 2024, pp. 1257-1266

DOI: https://doi.org/10.52436/1.jutif.2024.5.5.2043
p-ISSN: 2723-3863
e-ISSN: 2723-3871

IMPLEMENTATION OF DEEP LEARNING MODELS IN HATE SPEECH DETECTION ON TWITTER USING AN NATURAL LANGUAGE PROCESSING APPROACH

Muhammad Arifin[*1], Deni Mahdiana[*2]

[1]Computer Science Master's Study Program, Faculty of Information Technology, Universitas Budi Luhur, Indonesia
[2]Information System Study Program, Faculty of Information Technology, Universitas Budi Luhur, Indonesia
Email: [1]2311601518@student.budiluhur.ac.id, [2]deni.mahdiana@budiluhur.ac.id

(Article received: May 05, 2024; Revision: May 16, 2024; published: October 20, 2024)

Abstract

In the digital era, the misuse of the freedom to communicate on the internet often leads to problems such as the spread of hate speech, which can harm individuals based on race, religion, and other characteristics. This issue requires effective solutions for content moderation, particularly on social media platforms like Twitter. This research develops a deep learning model utilizing Natural Language Processing (NLP) to detect hate speech and aims to improve existing content moderation mechanisms. The methods used include data collection, preprocessing through techniques such as case folding, tokenization, lemmatization, and model creation using TensorFlow Extended (TFX) involving embedding, dense, and global pooling layers. The model is trained to optimize accuracy by minimizing the loss function and closely monitoring evaluation metrics. The results show that this model achieves a prediction accuracy of 84%, an AUC value of 0.796, and a binary accuracy of 76%. The conclusion of this research is that the use of deep learning and NLP in detecting hate speech offers a highly potential approach to enhancing digital content moderation, providing a solution that is not only efficient but also accurate.

Keywords: *Deep Learning, Hate Speech, Natural Language Processing, TensorFlow Extended.*

IMPLEMENTASI MODEL *DEEP LEARNING* DALAM DETEKSI UJARAN KEBENCIAN DI TWITTER DENGAN PENDEKATAN *NATURAL LANGUAGE PROCESSING*

Abstrak

Di era digital, penyalahgunaan kebebasan berkomunikasi di internet sering memunculkan masalah seperti penyebaran ujaran kebencian, yang dapat merugikan individu berdasarkan ras, agama, dan karakteristik lainnya. Masalah ini memerlukan solusi efektif untuk moderasi konten, khususnya di platform media sosial seperti Twitter. Penelitian ini mengembangkan model *deep learning* yang memanfaatkan *Natural Language Processing* (*NLP*) untuk mendeteksi ujaran kebencian dan bertujuan untuk memperbaiki mekanisme moderasi konten yang ada. Metode yang digunakan meliputi pengumpulan data, *preprocessing* melalui teknik *case folding*, tokenisasi, *lemmatization*, dan pembuatan model menggunakan *TensorFlow Extended* (*TFX*) yang melibatkan lapisan *embedding*, *dense*, dan *global pooling*. Model ini dilatih untuk mengoptimalkan akurasi dengan mengurangi fungsi loss dan memantau metrik evaluasi secara ketat. Hasilnya menunjukkan bahwa model ini mencapai akurasi prediksi sebesar 84%, nilai *AUC* 0.796, dan *binary accuracy* 76%. Kesimpulan dari penelitian ini adalah bahwa penggunaan *deep learning* dan *NLP* dalam mendeteksi ujaran kebencian menawarkan pendekatan yang sangat berpotensi untuk meningkatkan moderasi konten digital, dengan memberikan solusi yang tidak hanya efisien tetapi juga akurat.

Kata kunci: *Deep Learning, Natural Language Processing, TensorFlow Extended, ujaran kebencian.*

1. PENDAHULUAN

Di era digital saat ini, internet telah menjadi platform komunikasi global yang memberikan kebebasan bagi pengguna untuk mengekspresikan pendapat mereka secara bebas. Kebebasan dalam berkomunikasi ini telah menciptakan saluran yang memudahkan untuk berbagi informasi dan ide secara

instan [1]. Namun, kebebasan yang sama ini juga memiliki dampak buruk seperti penyebaran berita bohong, *Cyberbullying* dan ujaran kebencian[2], [3]. Ujaran kebencian biasanya dimaksudkan untuk mendorong tindakan kekerasan dan menghasilkan permusuhan terhadap individu atau kelompok oleh penggunaan bahasa yang menyerang, merendahkan, atau menghasut dengan sasaran seperti etnis, agama, gender, orientasi seksual, dan usia[4], [5], [6]. Ucapan yang mengandung kebencian berdampak pada semua individu, tanpa memperhatikan usia, baik itu anak-anak yang masih sangat muda, orang dewasa, maupun mereka yang telah lanjut usia [7].

Setiap hari, banyak orang menggunakan berbagai *platform* media sosial untuk menyampaikan pikiran, perasaan, dan perkembangan mereka. Sering kali, komentar yang diungkapkan cenderung bersifat negatif dan mengandung ujaran kebencian melebihi komentar positif, yang dapat menimbulkan berbagai masalah. Oleh karena itu, sangat penting untuk memilih metode yang efektif untuk mendeteksi dan menganalisis kata-kata tersebut, serta mengevaluasi hasilnya dengan akurat[8]. *Machine learning* dapat digunakan dalam mengidentifikasi ujaran kebencian, namun memerlukan jumlah data yang sangat banyak yang menjadi salah satu tantangan utamanya[9].

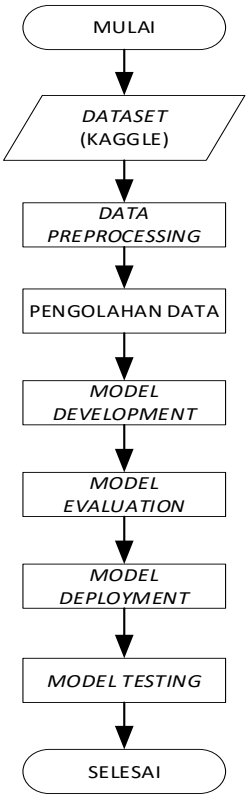
Dalam konteks klasifikasi, terdapat perbedaan kinerja antara metode *classical machine learning* dan *deep learning*. Metode *classical machine learning* memang menunjukkan peningkatan performa seiring dengan penambahan data, namun kinerjanya seringkali mencapai titik jenuh. Di sisi lain, metode *deep learning* terus menunjukkan peningkatan performa yang signifikan sejalan dengan peningkatan volume data yang diproses[10], [11]. Metode konvensional yang berbasis aturan atau kata kunci sering kali menghadapi kendala dalam mengenali secara tepat substansi dari ujaran kebencian, karena ujaran kebencian dapat bervariasi bentuknya seperti penggunaan bahasa yang tidak langsung, perubahan dalam slang, atau pernyataan yang sangat bergantung pada konteks[7], [12].

Pendeteksian ujaran kebencian dengan menggunakan teknik *deep learning*, terutam menggunakan pendekatan *Natural Language Processing* (*NLP*) mampu beradaptasi dengan perubahan konteks dan evolusi bahasa. *NLP* adalah cabang ilmu komputer yang berfokus pada pemahaman dan pemrosesan bahasa manusia oleh komputer[13]. Selain hanya *NLP* juga memperhatikan hierarki struktur bahasa[14]. Dengan kata lain, *NLP* mempertimbangkan bagaimana beberapa kata membentuk sebuah frase, beberapa frase membentuk sebuah kalimat, dan akhirnya, bagaimana kalimat-kalimat tersebut mengkomunikasikan pemikiran atau gagasan. Tujuan *NLP* adalah menemukan hubungan dengan kata-kata dan mengidentifikasi maknanya dari bahasa tersebut[15].

Tujuan dari penelitian ini adalah untuk mengimplementasikan teknologi *deep learning* dengan pendekatan *NLP* untuk mendeteksi tweet yang yang mengandung ujaran kebencian. Dengan pendekatan ini, diharapkan dapat dihasilkan model klasifikasi yang lebih akurat dan mampu menangani berbagai variasi kalimat dengan lebih efektif.

Penelitian ini diharapkan memberikan kontribusi signifikan dalam bidang pengolahan bahasa alami dan keamanan siber dengan menyediakan model yang lebih adaptif terhadap evolusi bahasa dan konteks sosial, sehingga mampu meningkatkan efisiensi dan akurasi moderasi konten digital di platform media sosial. Model ini juga diharapkan dapat menjadi dasar bagi pengembangan teknologi moderasi konten yang lebih canggih dan responsif, serta mendorong penelitian lanjutan untuk menghadapi tantangan dalam deteksi ujaran kebencian secara lebih efektif.

2. METODE PENELITIAN



Gambar 1. Alur Penelitian.

Alur penelitian dalam implementasi model *deep learning* dalam deteksi ujaran kebencian di twitter dengan pendekatan *NLP* terdiri dari beberapa tahapan, seperti terlihat pada Gambar 1. Tahap awal penelitian ini dimulai dengan pengumpulan *dataset*, dilanjutkan dengan *preprocessing data* menggunakan pendekatan *NLP* untuk meningkatkan kualitas dan relevansi data. Selanjutnya, data diolah melalui proses yang meliputi *data ingestion*, *data validation*, dan *data preprocessing* untuk memastikan

konsistensi dan akurasi. Setelah *dataset* siap dan terverifikasi, proses berlanjut ke tahap *model development*, di mana model dibangun dan disesuaikan untuk memenuhi spesifikasi penelitian. Model yang telah dilatih kemudian dievaluasi secara intensif untuk mengukur kinerjanya menggunakan berbagai metrik evaluasi. Setelah dinilai memiliki kinerja yang memadai dan memenuhi semua kriteria yang ditentukan, model tersebut kemudian di-*deploy*. Langkah terakhir adalah pengujian model yang telah di-*deploy* menggunakan program Python, yang bertujuan untuk memverifikasi kinerjanya dalam skenario dunia nyata dan memastikan bahwa model dapat diandalkan sebelum diimplementasikan secara luas.

2.1. *Dataset*

Pada penelitian ini, *dataset* Twitter yang dihasilkan oleh Kaggle yang berisi 150.000 *tweet* pengguna, yang terdiri dari 2 kolom yaitu text dan label. Tabel 1 menjelaskan informasi *dataset* yang didapat.

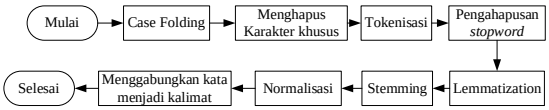
Tabel 1. Informasi *Dataset*

No	Column	Count	Data Type
0	text	150000	object
1	label	150000	int64

2.2. *Preprocessing* Data

Preprocessing data adalah langkah penting dalam mempersiapkan data teks untuk digunakan dalam model atau aplikasi. Langkah-langkah ini bertujuan untuk membersihkan, merapihkan, dan mengubah teks menjadi representasi yang lebih terstruktur sehingga dapat dimengerti oleh model *machine learning*. *Preprocessing* data dengan pendekatan *NLP* dapat membantu meningkatkan kualitas data dan memungkinkan model *NLP* untuk mempelajari pola-pola yang lebih bermakna dalam teks. *Preprocessing* terdiri dari *cese folding*, menghapus karakter khusus, tokenisasi, *lemmatization*, s*temming* dan mengembalikan kata ke bentuk semula, alur *preprocessing* dapat dilihat pada gambar 2. *Case folding* yaitu mengubah semua kalimat menjadi *lower case*. Menghapus karakter khusus dilakukan dengan menghapus tanda baca, angka dan karakter khusus lainnya.

Tokenisasi yaitu memotong urutan kata pada kalimat menjadi beberapa bagian yang disebut dengan token dan pada saat yang sama mungkin menghilangkan karakter tertentu misalnya tanda baca. Tahap *lemmatization* adalah proses mengubah kata-kata dalam teks ke bentuk dasar (lemmas), dengan mempertimbangkan konteks linguistik dan kategori gramatikal kata-kata tersebut. *Stemming* yaitu metode yang bertujuan untuk mencari bentuk dasar atau akar dari suatu kata dengan menghilangkan afiksnya. Setelah itu, kata tersebut diganti dengan bentuk dasarnya sehingga kata-kata yang memiliki akar yang sama dapat disatukan. Penggabungan kata dilakukan dengan menggabungkan kembali kata yang dihasilkan dari proses tokenisasi kembali ke bentuk semula.



Gambar 2. Alur *Preprocessing*

2.3. *Pengolahan Data*

Proses pengolahan data dilakukan dalam 3 tahapan yaitu *data ingestion*, *data validation*, *data preprocessing*. *Data ingestion* bertujuan untuk mengumpulkan data dari sumbernya ke dalam format yang dapat digunakan oleh model. Data diambil dari file *CSV* menggunakan komponen *CsvExampleGen*. Kemudian data dibagi menjadi dua set yaitu data *training* dan data evaluasi dengan rasio 80:20. Ini memastikan bahwa model memiliki data untuk belajar dan dievaluasi.

Tahap selanjutnya adalah data validation, setelah data diambil, tahap ini mengevaluasi data untuk memastikan kecocokannya dengan *schema* yang telah ditentukan sebelumnya. *Schema* tersebut mendefinisikan struktur data yang diharapkan, termasuk jenis fitur (misalnya teks atau kategori) dan jenis nilai yang diperbolehkan. komponen *StatisticsGen* dan *SchemaGen* digunakan untuk membuat statistik dan schema dari data. Kemudian, *ExampleValidator* digunakan untuk mendeteksi anomali atau kesalahan yang mungkin terjadi dalam data.

Tahap terakhir adalah *data preprocessing* (*Transform*), tahap di mana data mentah yang telah diambil dan divalidasi akan diubah menjadi format yang sesuai untuk proses pelatihan model. Proses transformasi ini melibatkan berbagai langkah untuk mengubah data menjadi representasi yang lebih sesuai untuk pemrosesan oleh model *machine learning*.

2.4. *Model Developement*

Pada tahap ini, didefinisikan arsitektur model yang akan digunakan dan melatihnya menggunakan data yang telah diproses. Pada penelitian ini arsitektur model yang digunakan adalah jaringan saraf tiruan sederhana yang terdiri dari beberapa lapisan. Arsitektur model ditunjukkan pada gambar 3. Total parameter yang dapat di-*train* dalam model tersebut adalah 163,201. Model ini menggunakan kombinasi layer *embedding*, *dense*, dan *global pooling* untuk memproses teks input dan menghasilkan prediksi klasifikasi biner. Model tersebut kemudian dilatih menggunakan data latihan dengan meminimalkan fungsi loss yang ditentukan dan memantau metrik evaluasi, seperti *binary accuracy*.

Layer (type)	Output Shape	Param #
text_xf (InputLayer)	[(None, 1)]	0
tf.reshape_6 (TFOpLambda)	(None,)	0
text_vectorization_6 (TextVectorization)	(None, 100)	0
embedding (Embedding)	(None, 100, 16)	160000
global_average_pooling1d_6 (GlobalAveragePooling1D)	(None, 16)	0
dense_18 (Dense)	(None, 64)	1088
dense_19 (Dense)	(None, 32)	2080
dense_20 (Dense)	(None, 1)	33

Total params: 163,201
Trainable params: 163,201
Non-trainable params: 0

Gambar 3. Arsitektur Model

2.5. *Model Evaluation*

Tahap *Model Evaluation* dilakukan setelah model dilatih dan dievaluasi menggunakan data yang dipisahkan antara *training* dan evaluasi. Tahap ini memanfaatkan *TensorFlow Model Analysis* (*TFMA*) untuk memberikan *insight* tentang kinerja model yang dihasilkan. Konfigurasi evaluasi didefinisikan pertama dengan *eval_config*, mencakup spesifikasi model yang dievaluasi, *slicing* untuk analisis data, dan metrik yang dievaluasi. Setelah itu, komponen *Evaluator* digunakan untuk mengevaluasi model hasil pelatihan dari *Trainer*. Komponen ini membutuhkan data contoh dari *CsvExampleGen*, model hasil pelatihan, dan model dasar dari *resolver*. Dengan menggunakan konfigurasi evaluasi yang telah ditentukan, Evaluator menghasilkan output evaluasi model, termasuk metrik yang telah ditentukan sebelumnya. Hasil evaluasi kemudian dirender dan divisualisasikan menggunakan *TensorFlow Model Analysis* (*TFMA*). Visualisasi ini membantu dalam pemahaman kinerja model, termasuk pemetaan metrik evaluasi terhadap *slice* data yang relevan.

Persamaan yang digunakan dalam evaluasi model meliputi beberapa metrik penting yaitu AUC dan *Binary Accuracy*. *AUC (Area Under the Curve)* mengukur kemampuan model dalam membedakan antara kelas positif dan negatif. Nilai *AUC* dihitung dari kurva *Receiver Operating Characteristic (ROC)*, yang menunjukkan hubungan antara *True Positive Rate (TPR)* dan *False Positive Rate (FPR)*. Nilai *AUC* berkisar antara 0 dan 1, dengan nilai mendekati 1 menunjukkan kemampuan model yang baik dalam membedakan antara kelas. Nilai *AUC* dihitung menggunakan persamaan 1.

$$AUC = \int_0^1 TPR\left(FPR^{-1}(x)\right)dx \quad (1)$$

Selain itu, *Binary Accuracy* mengukur proporsi prediksi yang benar di antara semua prediksi. *Binary Accuracy* dihitung dengan membandingkan jumlah prediksi benar (baik positif maupun negatif) dengan total prediksi, di mana *TP* (*True Positives*) adalah jumlah prediksi positif yang benar, *TN* (*True Negatives*) adalah jumlah prediksi negatif yang benar, *FP* (*False Positives*) adalah jumlah prediksi positif yang salah, dan *FN* (*False Negatives*) adalah jumlah prediksi negatif yang salah. *Binary Accuracy* memberikan gambaran umum tentang keakuratan model dalam membuat prediksi biner, dengan nilai mendekati 1 menunjukkan akurasi yang tinggi. Nilai *Binary Accuracy* dihitung menggunakan persamaan 2.

$$Binary\ Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (2)$$

2.6. *Model Deployment*

Setelah model dievaluasi dan dinyatakan layak, langkah-langkah berikutnya dimulai dengan komponen *Pusher*. Tugas utama *Pusher* adalah mengekstrak model yang telah dilatih beserta semua artefak terkait dari konteks *TensorFlow Extended* (*TFX*). Selanjutnya, model tersebut disimpan dalam format yang siap digunakan untuk inferensi atau prediksi. Penyimpanan dilakukan dalam direktori yang telah ditentukan sebelumnya, yaitu "*serving_model_dir/hate-detection-model*".

Langkah selanjutnya adalah menyiapkan *Dockerfile* dengan konfigurasi yang sesuai dan membangun *Docker image* dengan menggunakan perintah "*docker build -t hate-detection-tf-serving* .". Tindakan ini akan menciptakan *Docker image* yang berdasarkan pada *Docker file* yang telah disiapkan sebelumnya. *Docker image* ini akan berisikan *TensorFlow Serving* serta model yang telah disimpan di dalam direktori "*/models*", dengan nama "*model hate-detection-model*".

Ketika *Docker image* telah berhasil dibangun, langkah berikutnya adalah menjalankan *Docker container* dari *image* yang telah dibuat. Dapat dilakukan dengan menjalankan perintah "*docker run -p 8080:8501 hate-detection-tf-serving*". Tindakan ini akan menciptakan sebuah *container Docker* yang menjalankan *TensorFlow Serving* dengan model yang telah dimuat. Layanan *TensorFlow Serving* tersebut akan tersedia pada *port* 8080 di *host* yang digunakan.

2.7. *Model Testing*

Pengujian dimulai dengan pemeriksaan informasi model melalui *endpoint server* yang menjalankannya. Selanjutnya, *dataset* pengujian dimuat dan diproses. Setiap teks dalam *dataset* disiapkan menggunakan fungsi "*prepare_json*", yang mengonversi teks ke dalam format yang dapat diterima oleh model *TensorFlow*. Teks kemudian diposting ke *endpoint model* untuk mendapatkan prediksi. Prediksi dievaluasi dengan menggunakan *threshold* 0.6; jika nilai prediksi melebihi *threshold*, teks diklasifikasikan sebagai kebencian, jika tidak,

sebagai bukan kebencian. Hasil prediksi bersama dengan teks aslinya diekspor ke dalam file *CSV*"*predictions.csv*" untuk analisis lebih lanjut atau perbandingan dengan label yang sebenarnya.

3. HASIL DAN PEMBAHASAN

3.1. Dataset

Tabel 2 menampilkan beberapa contoh isi *dataset Tweets* yang diunduh dari kaggle yang selanjutnya akan dilakukan *preprocessing data* dengan pendekatan *NLP*.

Tabel 2. *Dataset Tweets*

No	Text	Label
0	denial of normal the con be asked to comment on tragedies an emotional retard	1
1	just by being able to tweet this insufferable bullshit proves trump a nazi you vagina	1
2	that is retarded you too cute to be single that is life	1
3	thought of a real badass mongol style declaration of war the attackers capture a citizen of the soon to be	1
4	afro american basho	1
...	...	...
149995	daniel barton is a year old pedophilia that loves little teenage girls he did years and months in the us navy before he what a slut dishonorable discharge he what a slut sent to miramar prison for molesting his stepdaughters and stealing his son identity	0
149996	science and nature journal looks like a solid and relevant source with a good reputation i think whatever can rely on it more than the others for this article nice research	0
149997	why cannot make erotic to a separate page erotic is an it means cool a k a used in many instances so i make it to a separate page for ease to search information about anything related to erotic don t you think so	0
149998	unique i have undone your edit to ok battery royal artillery where your edit summary what a slut unlinked unique no evidence for notability for this honor just to clarify unique is the only battle honor of the royal artillery royal horse artillery and the royal engineers the royal artillery what a slut present in nearly all battles and would have earned most of the honors awarded to cavalry and infantry regiments in william iv awarded the motto unique meaning in place of all battle honors	0
149999	use of capital in flood i changed instances where a capital what a slut used in flood i did not change red river flood as it should have consensus the same holds for red river flood in the united states	0

3.2. *Preprocessing* Data

Untuk meningkatkan efisiensi dan memudahkan proses pemodelan, data *tweet* yang sudah dikumpulkan akan menjalani beberapa langkah pra-pemrosesan. Langkah-langkah tersebut ditunjukkan dalam Tabel 3.

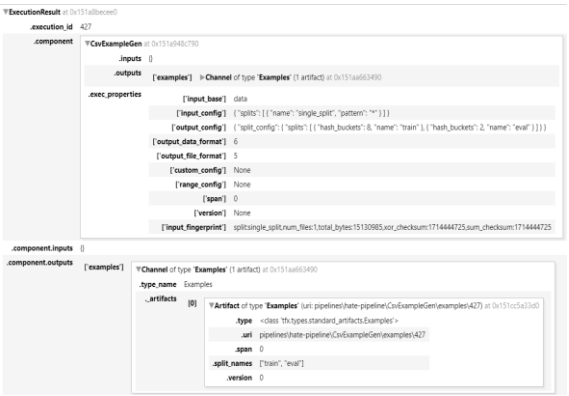
Tabel 3. Alur Pra-Pemrosesan Data

Proses	Text
Original	denial of normal the con be asked to comment on tragedies an emotional retard
Lowercase	denial of normal the con be asked to comment on tragedies an emotional retard
No punctuation	denial of normal the con be asked to comment on tragedies an emotional retard
Tokenized	['denial', 'of', 'normal', 'the', 'con', 'be', 'asked', 'to', 'comment', 'on', 'tragedies', 'an', 'emotional', 'retard']
Lemmatized	['denial', 'of', 'normal', 'the', 'con', 'be', 'asked', 'to', 'comment', 'on', 'tragedy', 'an', 'emotional', 'retard']
Stemmed	['denial', 'of', 'normal', 'the', 'con', 'be', 'ask', 'to', 'comment', 'on', 'tragedi', 'an', 'emot', 'retard']

3.3. Pengolahan Data

3.3.1. *Data Ingestion*

Data ingestion dilakukan dengan menggunakan komponen *CsvExampleGen()* yang disediakan oleh *TFX*. Komponen ini dipilih karena dataset yang kita gunakan memiliki format *CSV*. Pembagian *dataset* dilakukan untuk digunakan pada proses *training* ("*train*") dan *evaluasi* ("*eval*"). Pembagian ini dilakukan dengan mengatur parameter *output_config* dengan rasio 8:2. Hasil konfigurasi yang terdapat pada komponen *ExampleGen* ditunjukkan pada gambar 4.

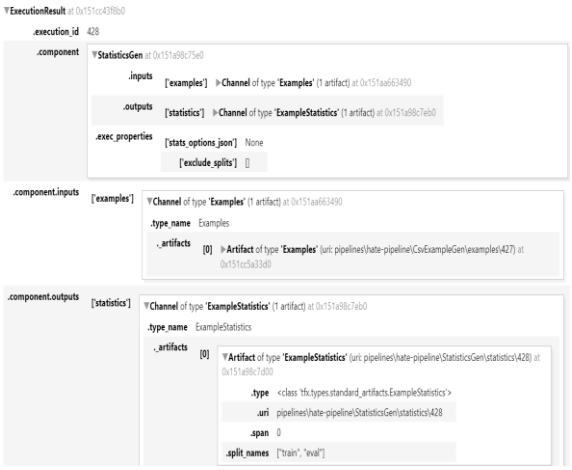


Gambar 4. Hasil Konfigurasi Komponen *ExampleGen*

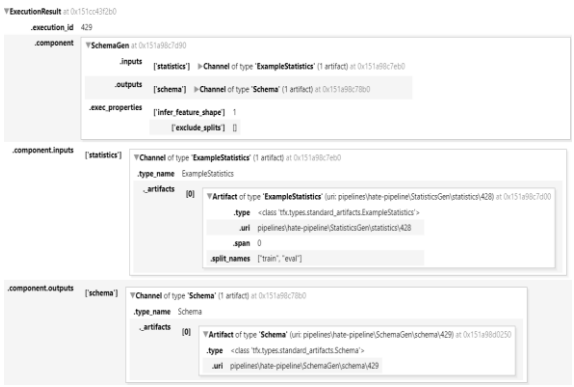
3.3.2. *Data Validation*

Proses data validation dilakukan dalam tiga tahapan yaitu membuat *summary statistic*, membuat *data schema*, mengidentifikasi anomali *dataset*. Summary statistic dibuat menggunakan komponen *statisticsGen()*, komponen ini memiliki parameter *input* berupa *example* untuk menerima *dataset* dari komponen *ExampleGen*. Hasil konfigurasi komponen *statisticGen* ditunjukkan pada gambar 5.

Setelah membuat *summary statistic*, langkah selanjutnya adalah membuat *data scema* menggunakan komponen *ScemaGen()*. Komponen ini akan menerima *input* berupa *summary statistic*. Hasil konfigurasi komponen *ScemaGen* ditunjukkan pada gambar 6.



Gambar 5. Konfigurasi Komponen StatisticGen



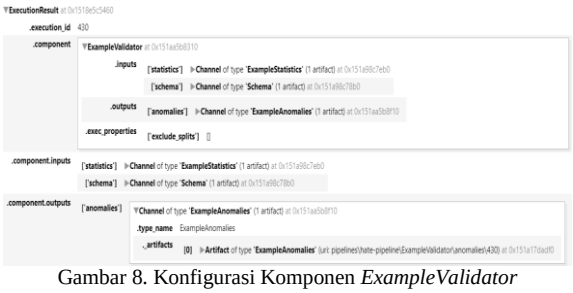
Gambar 6. Hasil Konfigurasi Komponen *ScemaGen*

Tampilan skema data dapat dilihat pada gambar 7, yang menyajikan visualisasi struktur dan tipe data dari kedua fitur tersebut dalam format yang mudah dipahami. Terlihat dari *SchemaGen* yang telah dibuat, terdapat dua fitur, yaitu label dan teks. Label di sini untuk menentukan prediksi apakah teks merupakan kalimat yang menandakan ujaran kebencian atau tidak, sementara teks merupakan masukan untuk menentukan label. Label bertipe *integer* dan teks bertipe *byte*, dan masing-masing fitur tersebut diperlukan untuk melakukan prediksi.

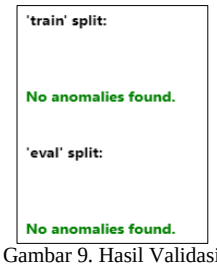
Feature name	Type	Presence	Valency	Domain
'label'	INT	required		-
'text'	BYTES	required		-

Gambar 7. Tampilan *Data Schema*

Tahap terakhir adalah mengidentifikasi anomali pada *dataset*. Proses ini dilakukan menggunakan komponen *ExampleValidator()* yang telah disediakan oleh TFX. Komponen ini membutuhkan keluaran dari *statistics_gen* dan *schema_gen* sebagai parameter *input*. Detail komponen *ExampleValidator* ditunjukkan pada gambar 8.



Gambar 8. Konfigurasi Komponen *ExampleValidator*



Gambar 9. Hasil Validasi

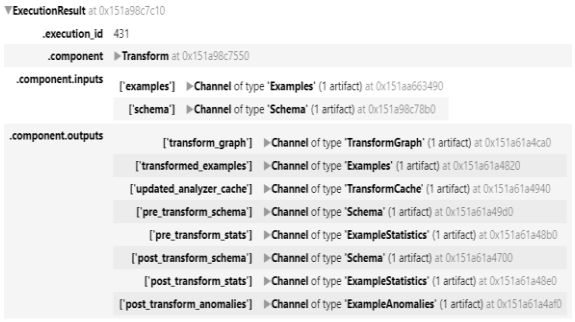
Bersarkan hasil validasi, tidak terdapat anomali pada *dataset* yang digunakan.

3.3.3. *Data Preprocessing* (*Transform*)

Setelah melakukan data validation, tahap berikutnya adalah melakukan data preprocessing. Tujuan dari tahap ini adalah mengubah data mentah menjadi data yang siap digunakan untuk melatih model. *TensorFlow Transform* dan komponen *Transform* digunakan untuk melakukan data *preprocessing*. Sebuah modul bernama *hate_transform.py* yang mengandung dua fungsi penting: *transformed_name()* dan *preprocessing_fn()*. Fungsi *transformed_name()* bertujuan untuk mengubah nama fitur yang telah di-*transform* dengan menambahkan "_xf" pada nama fitur asli.

Fungsi *preprocessing_fn()* digunakan untuk menerapkan langkah-langkah *preprocessing* yang sederhana pada *dataset*. Dalam fungsi ini, data pada fitur "*text*" diubah menjadi bentuk huruf kecil (*lowercase*) untuk standarisasi, dan data pada fitur "*label*" dikonversi menjadi format bilangan bulat (tf.int64) untuk memastikan konsistensi tipe data.

Setelah modul yang berisi fungsi-fungsi *preprocessing* ini dibuat, langkah selanjutnya adalah mendefinisikan komponen *Transform()* dalam *pipeline TensorFlow Extended* (*TFX*). Komponen ini diatur untuk menerima *input* berupa *examples* yang berisi *dataset* dari *ExampleGen*, *schema* yang berisi skema data dari *SchemaGen*, serta *module_file* yang merujuk pada berkas modul yang telah dibuat sebelumnya. Dengan demikian, komponen *Transform()* ini dapat menjalankan fungsi *preprocessing* secara efektif berdasarkan logika yang telah didefinisikan dalam modul tersebut. Konfigurasi komponen Transform ditunjuukan pada gambar 10.



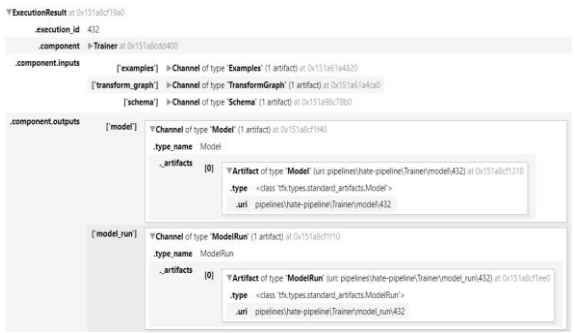
Gambar 10. Konfigurasi Komponen Transform.

3.4. *Model Developement*

Dalam tahap developement langkah yang harus dilakukan adalah membuat dan pelatihan model menggunakan komponen *Trainer*. Komponen ini memerlukan beberapa input termasuk sebuah file modul yang berisi fungsi pelatihan. Modul *hate_trainer.py* dikembangkan yang mencakup berbagai fungsi seperti *transformed_name()* untuk mengubah nama fitur yang telah ditransformasi, *gzip_reader_fn()* untuk memuat data dalam format *TFRecord*, *input_fn()* untuk memuat dan membagi data ke dalam *batch*, serta *model_builder()* yang bertanggung jawab atas pembuatan arsitektur model. Model ini menggunakan lapisan *embedding* yang diunduh melalui *TensorFlow Hub* dan diintegrasikan dengan operasi lain seperti *pooling* dan *densification* untuk pemrosesan sinyal teks.

Pada komponen *Trainer*, file modul yang telah dibuat tadi dimasukkan sebagai salah satu *input*-an, di samping itu juga menerima *dataset* dari komponen *ExampleGen*, skema data dari *SchemaGen*, dan grafik transformasi dari komponen *Transform*. Tambahan lain adalah parameter pelatihan dan evaluasi yang diatur melalui *TrainArgs* dan *EvalArgs* untuk mengendalikan proses pelatihan dan evaluasi model.

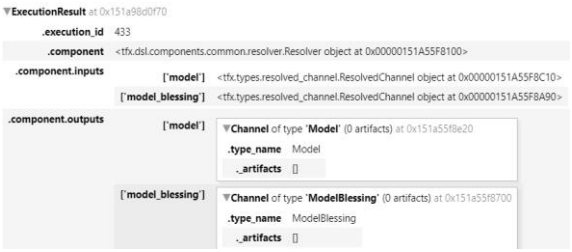
Pelaksanaan pelatihan model dilakukan dalam lingkungan *TensorFlow Extended* (*TFX*) yang memungkinkan untuk proses yang konsisten dan terukur. Pelatihan ini mencakup *callback* seperti *TensorBoard* untuk visualisasi proses pelatihan, *EarlyStopping* untuk menghentikan pelatihan ketika tidak terjadi peningkatan, dan *ModelCheckpoint* untuk menyimpan model terbaik. Detail konfigurasi komponen *trainer* ditunjukkan pada gambar 11.



Gambar 11. Konfigurasi Komponen *Trainer*

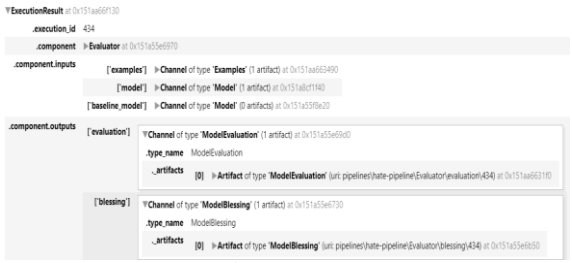
3.5. *Model Evaluation*

Setelah proses pengembangan model, tahap selanjutnya adalah membuat tahapan analisis dan validasi model. Untuk menjalankan tahapan ini perlu didefinisikan dua buah komponen yaitu komponen *resolver* dan komponen *evaluator*. Komponen resolver digunakan untuk membandingkan beberapa versi model yang berbeda. Konfigurasi komponen *resolver* ditunjukkan pada gambar 12.



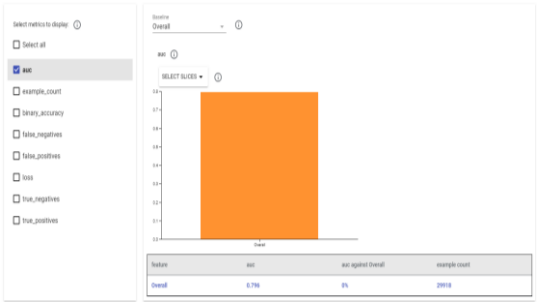
Gambar 12. Konfigurasi Komponen *Resolver*

Setelah mendefinisikan komponen *resolver*, tahap selanjutnya adalah membuat beberapa konfigurasi untuk mengevaluasi model menggunakan *library TFMA*. Setelah membuat konfigurasi yang akan digunakan untuk mengevaluasi model, tahap selanjutnya adalah mendefinisikan komponen *evaluator*. Komponen ini akan menerima beberapa inputan seperti *examples*, *model*, *baseline_model* dan *eval_config*. Hasil konfigurasi komponen *evaluator* ditunjukkan pada gambar 13.



Gambar 13. Konfigurasi Komponen *Evaluator*

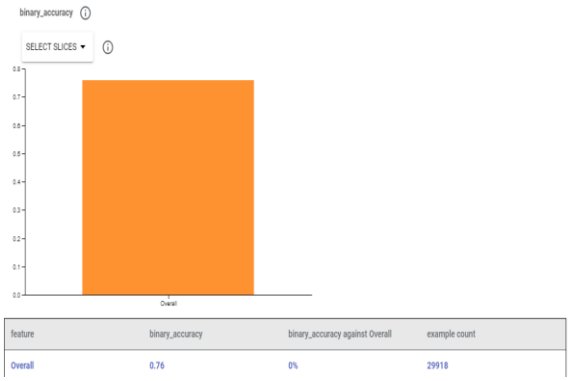
Hasil evaluasi dari komponen *evaluator* divisualisasikan menggunakan *library TFMA*.



Gambar 14. Nilai AUC

Hasil evaluasi model tersebut menunjukkan bahwa model tersebut memiliki performa yang cukup baik dengan nilai *AUC* (*Area Under the Curve*) sebesar 0.796 dan *binary accuracy* sebesar 76%. Nilai *AUC* yang relatif tinggi mengindikasikan bahwa model memiliki kemampuan yang efektif dalam

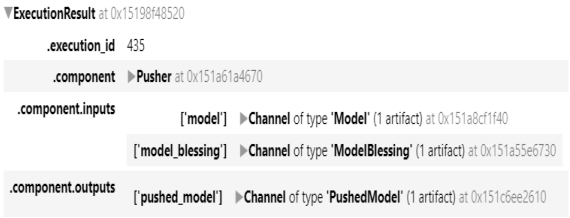
membedakan antara kelas positif dan negatif, sedangkan tingkat akurasi yang solid menunjukkan keakuratan prediksi yang baik dalam klasifikasi biner.



Gambar 15. Nilai *Binary Accuracy*

3.6. *Model Deployment*

Langkah pertama dalam deploy model adalah menambahkan komponen *Pusher()* ke dalam *pipeline* yang sebelumnya telah dibuat. Komponen ini akan menerima inputan berupa *trained model*, hasil evaluasi dari komponen *Evaluator*, dan argumen terkait *serving file path*. Konfigurasi komponen *pusher* dapat dilihat pada gambar 16.



Gambar 16. Konfigurasi Komponen *Pusher*

Setelah komponen *pusher* dijalankan maka akan muncul sebuah folder bernama *serving_model_dir* yang diakses oleh *TF-Serving* untuk mengambil model terbaru dan menjalankan di *production environment*. Langkah selanjutnya adalah membuat *docker file* untuk membuat *docker image*. Langkah terakhir adalah menjalankan *docker image* yang telah dibuat. Versi model yang digunakan oleh *TF-Serving* dapat dilihat pada gambar 18.

```
FROM tensorflow/serving:latest

COPY ./serving_model_dir /models
ENV MODEL_NAME=hate-detection-model
```

Gambar 17. *Docker File*

```
import requests
from pprint import PrettyPrinter

pp = PrettyPrinter()
pp.pprint(requests.get("http://localhost:8080/v1/models/hate-detection-model").json())
```

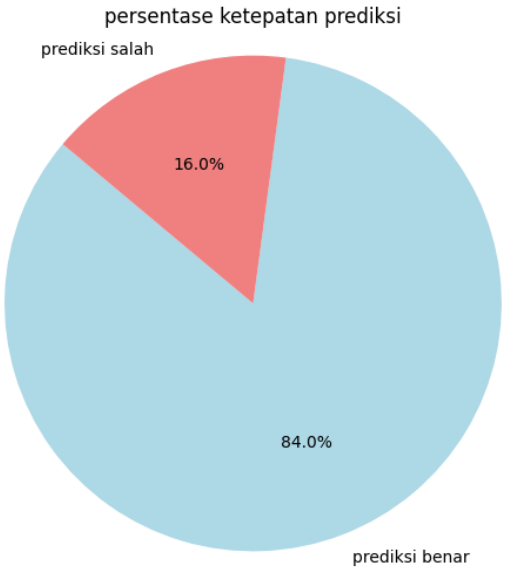
✓ 0.1s

```
{'model_version_status': [{'state': 'AVAILABLE',
                           'status': {'error_code': 'OK', 'error_message': ''},
                           'version': '1714360036'}]}
```

Gambar 18. Versi Model

3.7. *Model Testing*

Dalam pengujian ketepatan model dalam mengklasifikasikan *teweets* yang mengandung *Hate Speech* digunakan 100 *tweets* yang telah dilakukan *data preprocessing*. Pengujian dilakukan menggunakan program sederhana yang dibuat dengan bahasa pemrograman python.



Gambar 19. persentase ketepatan prediksi

Dari hasil pengujian, sebanyak 84% dari jumlah data training menunjukkan prediksi benar. Hasil tersebut menunjukkan bahwa model yang dibuat memiliki kinerja yang baik dalam mendeteksi *tweet* yang mengandung *hate speech*.

4. DISKUSI

Penelitian ini menunjukkan bahwa pemrosesan awal dengan pendekatan *NLP* dapat meningkatkan efisiensi dan memudahkan proses pemodelan. Hal ini didukung oleh hasil penelitian sebelumnya oleh Tabassum A, dkk., yang menyebutkan bahwa kalimat atau teks yang tidak terstruktur pada dasarnya sulit untuk dikonversi ke format yang dapat dimengerti mesin dan kesalahan apapun dalam tahap *preprocessing* dapat berdampak pada tahap selanjutnya[16].

Selanjutnya, proses pelatihan menggunakan model jaringan saraf tiruan dengan menerapkan teknik *EarlyStopping* untuk menghentikan pelatihan ketika tidak terjadi peningkatan, dan *ModelCheckpoint* untuk menyimpan model terbaik. mengacu pada penelitian yang dilakukan oleh bimantoro, dkk., penerapan *EarlyStopping* dan *ModelCheckpoint* akan mempercepat proses training dan dapat menghasilkan model dengan akurasi terbaik[17], [18].

Hasil evaluasi menunjukkan bahwa model memiliki performa yang cukup baik dengan nilai *binary accuracy* sebesar 75% dan dari hasil uji coba model dengan menggunakan program sederhana

yang telah dibuat menunjukkan hasil ketepatan prediksi sebesar 84%. Hal ini menunjukkan adanya konsistensi dalam performa model yang baik di lingkungan pengujian dan pengoperasian nyata. Selisih antara akurasi pengujian dan akurasi operasional dapat mengindikasikan bahwa model mungkin lebih efektif dalam mengatasi data atau situasi yang lebih dekat dengan kondisi nyata daripada yang terdapat dalam dataset pengujian yang mungkin lebih terbatas atau kurang variatif.

Hasil penelitian ini juga menunjukkan bahwa penggunaan teknik *embedding* melalui *TensorFlow Hub* memberikan kontribusi signifikan terhadap kemampuan model dalam memahami dan memproses sinyal teks. Pendekatan ini memungkinkan model untuk menangkap konteks dan hubungan antar kata yang lebih kompleks dibandingkan dengan metode konvensional. Dalam studi yang dilakukan oleh Doris Xin, dkk., penggunaan *embedding* terbukti meningkatkan akurasi dan efektivitas model dalam tugas-tugas pemrosesan bahasa alami[19].

Selain itu, penelitian ini membuktikan bahwa *pipeline TFX* yang diimplementasikan mampu mengotomatisasi dan menyederhanakan berbagai tahap dalam siklus hidup pengembangan model. Hal ini mencakup mulai dari *data ingestion*, validasi, *preprocessing*, pelatihan, hingga *deployment* model. Proses otomatisasi ini tidak hanya meningkatkan efisiensi, tetapi juga konsistensi dan *reproducibility* dari hasil model yang dikembangkan, sebagaimana didiskusikan dalam studi oleh Y. Zhou, dkk.[20]

Namun, penelitian ini juga mengakui adanya beberapa keterbatasan. Salah satunya adalah variasi dalam dataset yang digunakan. Meski dataset besar diunduh dari Kaggle, kualitas dan keragaman data tetap menjadi faktor penting yang dapat mempengaruhi performa model. Penelitian lanjutan disarankan untuk menguji model ini pada dataset lain yang lebih bervariasi untuk mengukur generalisasi model secara lebih luas.

5. KESIMPULAN

Penelitian ini berhasil mengimplementasikan model deep learning untuk deteksi ujaran kebencian di *twitter* menggunakan pendekatan *Natural Language Processing* (*NLP*). Proses pengembangan model meliputi beberapa tahapan kritis yang meliputi pengumpulan dan *preprocessing data*, pembuatan dan validasi model, serta pengujian akhir. *Preprocessing* data terbukti sangat penting untuk mengoptimalkan kinerja model dengan transformasi seperti *case folding*, tokenisasi, dan *lemmatization* yang meningkatkan struktur dan kualitas data masukan. Selanjutnya, data tersebut diolah melalui komponen-komponen *TensorFlow Extended* (*TFX*) untuk pengolahan data lebih lanjut, validasi, dan persiapan model untuk pelatihan.

Dalam tahap pengembangan model, teknik *deep learning* yang digunakan adalah jaringan saraf tiruan yang terdiri dari lapisan *embedding*, *dense*, dan *global pooling*. Model ini dilatih dengan menggunakan data yang telah diproses dan divalidasi, menghasilkan performa yang cukup baik dengan nilai *AUC* 0.796 dan *binary accuracy* 76%. Penerapan teknik *EarlyStopping* dan *ModelCheckpoint* dalam pelatihan membantu dalam optimisasi proses pelatihan, menghasilkan model yang tidak hanya akurat tetapi juga efisien. Evaluasi model menggunakan *TensorFlow Model Analysis* (*TFMA*) memberikan wawasan mendalam tentang kinerja model, memungkinkan penyesuaian dan peningkatan yang berbasis data.

Hasil pengujian model menunjukkan konsistensi performa yang baik dalam lingkungan nyata dengan ketepatan prediksi mencapai 84%. Ini menegaskan keefektifan model dalam mengidentifikasi ujaran kebencian di *Twitter*, mencerminkan adaptasi yang baik terhadap varian data nyata dibandingkan dengan *dataset* pelatihan. Hasil ini juga menunjukkan pentingnya tahap *preprocessing* dalam meningkatkan akurasi model. Kesimpulannya, penelitian ini menunjukkan potensi aplikasi *NLP* dalam pengembangan alat otomatis untuk monitoring media sosial, dengan implikasi signifikan untuk peningkatan moderasi konten dan keamanan digital.

DAFTAR PUSTAKA

[1] A. P. J. Dwitama and S. Hidayat, "Identifikasi Ujaran Kebencian Multilabel Pada Teks Twitter Berbahasa Indonesia Menggunakan Convolution Neural Network," *Jurnal Sistem Komputer dan Informatika (JSON)*, vol. 3, no. 2, p. 117, Dec. 2021, doi: 10.30865/json.v3i2.3610.

[2] I. Riadi, A. Fadlil, and Murni, "Identifying Hate Speech in Tweets with Sentiment Analysis on Indonesian Twitter Utilizing Support Vector Machine Algorithm," *Jurnal Ilmu Komputer dan Informatika*, vol. 9, no. 2, pp. 179–191, 2023.

[3] C. Erico, R. Salim, and D. Suhartono, "A Systematic Literature Review of Different Machine Learning Methods on Hate Speech Detection," *International Journal On Informatics Visualization*, vol. 4, no. 4, pp. 213–218, 2020.

[4] H. Xie, M. Namvar, and M. Risius, "A Review of Hate Speech Detection: Challenges and Innovations," in *AIS Electronic Library (AISeL)*, Haiderabad, Dec. 2023, pp. 1–9.

[5] C. Kumar, R. Kumar Yadav, and V. Namdeo, "A Review on Hate Speech Recognition on Social Media," *International Journal of Innovative Research in Technology and Management*, vol. 4, no. 4, pp. 50–57, 2020, [Online]. Available: www.ijirtm.com

[6] M. Amien Ibrahim *et al.*, "An Explainable AI

Model for Hate Speech Detection on Indonesian Twitter," *CommIT Journal*, vol. 16, no. 2, pp. 175–182, 2022.

[7] A. Zozan Miran and A. Mohsin Abdulazeez, "Detection of Hate-Speech Tweets Based on Deep Learning: A Review," *JISA (Jurnal Informatika dan Sains)*, vol. 6, no. 2, pp. 174–188, 2023.

[8] S. Rohmah and H. Kapi, "Analysis of Hate Speech Detection based on Obstacles and Solutions using Deeplearning Methods," *International Journal of Informatics Technology (INJIT)*, vol. 1, no. 3, pp. 108–117, 2023.

[9] Preethi and R. Dodmane, "Hate and Sarcasm Classification Using Machine Learning and Deep Learning Techniques: A Survey," *International Journal of Research in Engineering, Science and Management*, vol. 5, no. 7, pp. 24–30, Jul. 2022.

[10] S. Kurniawan and I. Budi, "Utilizing Translation to Enhance NLP Models in Offensive Language and Hate Speech Identification," *Journal of Engineering Sciences (Improsci)*, vol. 1, no. 4, pp. 182–197, Feb. 2024.

[11] A. Z. Miran and H. S. Yahia, "Hate Speech Detection in Social Media (Twitter) Using Neural Network," *Journal of Mobile Multimedia*, vol. 19, no. 3, pp. 765–798, 2023, doi: 10.13052/jmm1550-4646.1936.

[12] J. N. Saeed and A. M. Abdulazeez, "Facial Beauty Prediction and Analysis Based on Deep Convolutional Neural Network: A Review," *Journal of Soft Computing and Data Mining*, vol. 2, no. 1, pp. 1–12, Apr. 2021, doi: 10.30880/jscdm.2021.02.01.001.

[13] B. Li, Y. Hou, and W. Che, "Data augmentation approaches in natural language processing: A survey," *AI Open*, vol. 3, pp. 71–90, Jan. 2022, doi: 10.1016/j.aiopen.2022.03.001.

[14] A. Majid, D. Nugraha, and F. Dharma Adhinata, "Sentiment Analysis on Tiktok Application Reviews Using Natural Language Processing Approach," 2023, doi: 10.26858/jessi.v4i1.41897.

[15] A. P. R. Nababan, A. S. M. Lumenta, Y. D. Y. Rindengan, F. J. Pontoh, and Y. V. Akay, "Analisis Sentimen Twitter Pasca Pengumuman Hasil Pilpres 2019 Menggunakan Metode Lexicon Analysis," *Jurnal Teknik Informatika*, vol. 15, no. 1, pp. 33–34, Jan. 2020.

[16] A. Tabassum and R. R. Patil, "A Survey on Text Pre-Processing & Feature Extraction Techniques in Natural Language Processing," *International Research Journal of Engineering and Technology*, vol. 7, no. 6, pp. 4864–4867, 2020, [Online]. Available: www.irjet.net

[17] P. K. Roy, A. K. Tripathy, T. K. Das, and X. Z. Gao, "A framework for hate speech detection using deep convolutional neural network," *IEEE Access*, vol. 8, pp. 204951–204962, 2020, doi: 10.1109/ACCESS.2020.3037073.

[18] M. Z. Bimantoro and A. W. R. Emanuel, "Sheep Face Classification using Convolutional Neural Network," in *3rd 2021 East Indonesia Conference on Computer and Information Technology, EIConCIT 2021*, Institute of Electrical and Electronics Engineers Inc., Apr. 2021, pp. 111–115. doi: 10.1109/EIConCIT50028.2021.9431933.

[19] D. Xin, H. Miao, A. Parameswaran, and N. Polyzotis, "Production Machine Learning Pipelines: Empirical Analysis and Optimization Opportunities," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Association for Computing Machinery, 2021, pp. 2639–2652. doi: 10.1145/3448016.3457566.

[20] Y. Zhou, Y. Yu, and B. Ding, "Towards MLOps: A Case Study of ML Pipeline Platform," in *Proceedings - 2020 International Conference on Artificial Intelligence and Computer Engineering, ICAICE 2020*, Institute of Electrical and Electronics Engineers Inc., Oct. 2020, pp. 494–500. doi: 10.1109/ICAICE51518.2020.00102.