BAB III
METODOLOGI PENELITIAN

3.1 Analisis sistem

Analisis sistem bertujuan untuk memberikan pendalaman penuh tentang sistem yang akan dibuat dan memahami permasalahan yang ada dalam mengembangkan sebuah sistem yang akan dibuat. Pengecekan tulisan tangan biasanya dilakukan secara manual dengan membaca huruf satu persatu. Hal tersebut mempunyai banyak kekurangan mulai dari waktu yang dibutuhkan hingga hasil yang kurang akurat.

Dalam mengenali kata atau kalimat, manusia menggunakan pengetahuannya untuk mengenali huruf kemudian merangkai huruf-huruf tersebut. cara kerja seperti ini di akomodir oleh sistem OCR, di mana OCR dibangun menggunakan serangkaian algoritma dalam memproses citra kemudian melakukan pengenalan dengan membandingkan huruf-huruf tersebut dengan *dataset* yang dimilikinya. Dalam membandingkan proses pengenalan teks manusia dengan teknologi OCR (*Optical Character Recognition*), terdapat dua hipotesis yang relevan. Pertama, hipotesis menyatakan bahwa manusia dan OCR memiliki proses dasar yang serupa dalam mengenali teks. Seperti OCR, manusia juga mengandalkan pengenalan pola untuk mengonversi teks menjadi makna yang dapat dipahami. Meskipun dengan pendekatan yang berbeda, manusia menggunakan otak untuk mengenali huruf, kata, dan kalimat, sedangkan OCR menggunakan algoritma dan pemrosesan citra. Hipotesis kedua mengemukakan bahwa konteks dan pengalaman mempengaruhi kesesuaian kata yang dipilih oleh manusia dan OCR. Manusia seringkali memanfaatkan konteks dan pengalaman mereka untuk memahami teks, sementara OCR mungkin hanya memilih kata berdasarkan karakter yang terlihat serupa tanpa mempertimbangkan konteks. Ini menimbulkan dugaan bahwa manusia cenderung lebih tepat dalam memilih kata yang sesuai dengan konteks, sedangkan OCR mungkin mengalami keterbatasan dalam hal ini. Potensi penelitian empiris dapat menguji dua hipotesis ini, memberikan pemahaman lebih lanjut tentang kesamaan dan perbedaan antara kemampuan visual manusia dengan teknologi OCR dalam konteks pengenalan dan pencocokan kata.

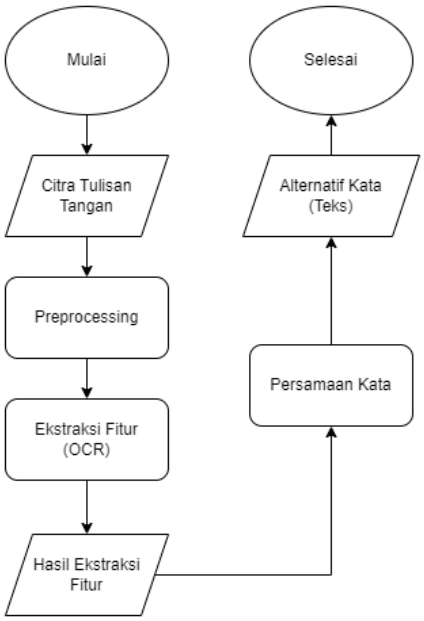
untuk dapat mengenali rangkaian huruf menjadi kata, manusia perlu mencari kata yang paling mirip dengan huruf yang sedang dibaca (dieja). mekanisme ini dilakukan dengan mencocokkan tiap huruf beserta urutannya dengan kata yang dipikirkan. Mekanisme ini sangat mirip dengan cara kerja *hamming distance*, di mana metode ini

banyak digunakan untuk mengukur kemiripan karakter *string*. Penggunaan *Hamming Distance* untuk mengukur jarak pencocokan kata dengan manusia memiliki korelasi positif dengan keberhasilan pemahaman teks. Dengan asumsi bahwa semakin rendah *Hamming Distance* antara kata yang diberikan oleh manusia dan kata sebenarnya, semakin baik pemahaman teks oleh manusia. Dengan melakukan pengujian yang melibatkan berbagai kata kepada responden manusia, lalu mengukur *Hamming Distance* antara kata yang mereka berikan dengan kata sebenarnya, dapat memberikan indikasi seberapa baik *Hamming Distance* memprediksi tingkat kesesuaian kata manusia dalam konteks pengenalan teks. Potensi aplikasi hipotesis ini adalah untuk pengembangan metode evaluasi yang lebih baik dalam bidang OCR, pemrosesan bahasa alami, dan pemahaman teks manusia.

Dengan menggunakan metode pengolahan citra digital yaitu OCR dan pengenalan kata *hamming distance*, mulai dari pengecekan tulisan bisa dimudahkan dengan hanya memasukkan gambar ke dalam aplikasi dan secara cepat menghasilkan hasil yang terbaik. Dari hal tersebut dibuatlah sistem pengecekan tulisan tangan menggunakan metode ekstraksi *Optical Character Recognition* (OCR) dan metode Persamaan *Hamming Distance*.

3.2 Perancangan Sistem

Perancangan sistem bertujuan untuk memberikan gambaran secara umum tentang sistem yang akan dibuat sehingga kebutuhan sistem dapat diketahui sebelumnya. Fungsi *flowchart* ialah memberi gambaran tentang program yang akan dibuat pada penelitian ini. Dalam proses rekognisi dan pelengkap kata tulisan tangan terdapat beberapa proses yang perlu dilalui. **Gambar 3.1** memperlihatkan cara alur kerja sistem atau *Flowchart* sistem secara visual dari awal hingga akhir.



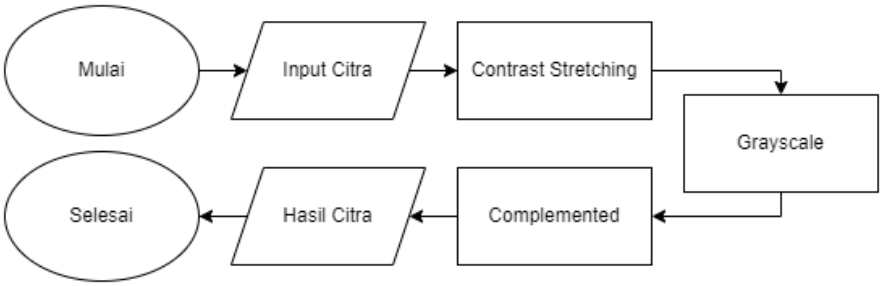
Gambar 3.1 Flowchart perancangan sistem

Gambar 3.1 menjelaskan proses sistem pengecekan tulisan tangan yang dilakukan dengan beberapa tahapan, berawal dari memasukkan data awal citra untuk setelahnya dilakukan *preprocessing* atau pemrosesan awal menggunakan *contrast stretching, grayscale* dan *complement*. Dilanjut dengan proses pengenalan citra menggunakan *Optical Character Recognition* (OCR). Hasil OCR berupa teks (karakter) yang digunakan sebagai data masukan pada *google colab*. Teks masukan pada *google colab* tersebut kemudian dilakukan pembersihan hasil dari karakter yang tidak diinginkan seperti karakter selain huruf. Setelah dibersihkan, dilakukan proses pencarian kata untuk menemukan kata yang kurang cocok hingga cocok menggunakan metode *Hamming Distance*. Setelah dilakukan pengecekan akan didapat sebuah hasil yang menunjukkan kata mana yang lebih cocok dari tulisan yang dicari dengan tingkat akurasi tiap kata yang ditemukan dengan kata awal.

Sebelum *preprocessing*, pengambilan data tulisan tangan pada penelitian ini menggunakan kamera poco m3 pro 5G dengan kamera utama dengan 48 *megapiksel* yang berjumlah 60 tulisan tangan dengan ekstensi .jpg. Tulisan tangan dilakukan oleh orang dengan usia 10-15 sebanyak 20 data, usia 20-25 sebanyak 20 data dan usia 50-55 sebanyak 20 data. Adapun dalam pembuatan sistem ini menggunakan perangkat lunak *Matlab* dengan Bahasa "*Matlab*", walaupun *Matlab* merupakan sebuah nama perangkat lunak tetapi Bahasa yang digunakan juga dinamai dengan nama "*Matlab*"(Moler & Little, 2020). Serta menggunakan perangkat lunak *Google colab* dengan Bahasa *python*.

3.2.1 *Preprocessing* atau Pemrosesan awal

Preprocessing adalah proses di mana citra di rubah untuk menyesuaikan format citra yang digunakan menjadi lebih sederhana untuk dilakukan analisis lebih lanjut. Dalam penelitian ini untuk Bagian *Preprocessing* terdiri dari beberapa metode untuk urutannya akan dijelaskan dengan *Flowchart* di bawah ini.



Gambar 3.2 Flowchart Tahapan Preprocessing

Gambar 3.2 menjelaskan Langkah-langkah pra-pemrosesan untuk penelitian ini. Penjelasan mengenai tahapan pengerjaannya pra-pemrosesan adalah sebagai berikut.

Pertama, dilakukan perbaikan kualitas masukan citra berupa peregangan kontras atau *contrast stretching*. Fungsi tersebut mampu meningkatkan kontras gambar sehingga menjadi lebih jelas.

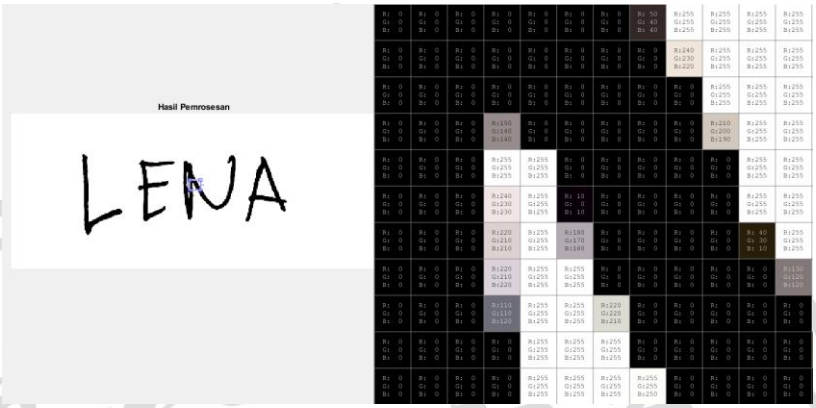
Gambar 3.3 Sebelum contrast (a) dan sesudah contrast stretching (b)

Dari **Gambar 3.3** memberikan gambaran hasil dari *imadjust*. *imadjust* akan mengubah nilai intensitas citra masukan menjadi lebih cerah karena intensitas citra masukan awal akan di tingkatkan pada rentang nilai intensitas keluaran.

Peninggian kontras memiliki intensitas tiga warna atau RGB yaitu merah (*red*), hijau (*green*) dan biru (*blue*). Untuk itu dibutuhkan 2x3 matriks yaitu matriks RGB untuk bagian *low-in* dan juga *high-in*. dalam penelitian ini menggunakan 0.9 untuk *low-in* disama ratakan untuk semua warna RGB dan 1 untuk *high-in* disama

ratakan untuk semua warna RGB dengan itu menghasilkan warna yang cocok untuk menaikkan intensitas warna objek hitam.

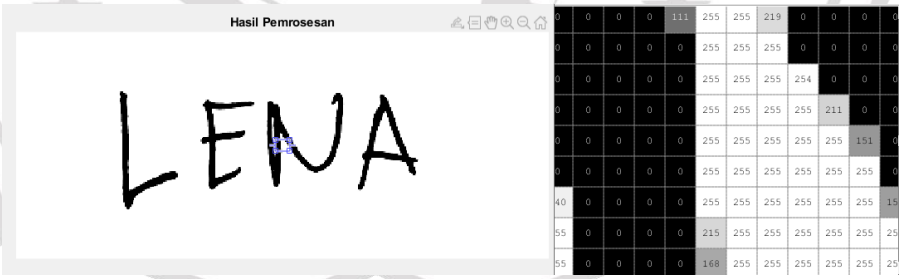
Setelah dilakukan proses peregangan kontras dihasilkan citra *Grayscale*. *Grayscale* adalah proses di mana citra akan diubah mode warnanya dari misal RGB ke warna ke abuan. Di bawah ini akan diberikan contoh gambar dengan mode warna RGB.



Gambar 3.4 Citra RGB dan Kanal Penyusunnya

Untuk mengubah menjadi *Grayscale* dilakukan perhitungan dengan persamaan sederhana dengan contoh nilai rata-rata kecerahan dalam tiga warna RGB yang berbeda menggunakan rumus persamaan (2.1).

Hasil dari proses *contrast stretching* berupa citra dalam bentuk *grayscale* di mana citra dalam bentuk *grayscale* ini dibutuhkan untuk mempermudah proses gambar dalam proses analisis lebih lanjut karena hanya memiliki satu nilai kanal pada setiap pikselnya yang memiliki nilai di antara 0-255.



Gambar 3.5 Citra Grayscale dengan nilai piksel

Selanjutnya dilakukan proses *complement* di mana proses ini akan membalikkan nilai intensitas dengan cara mengurangi nilai tertinggi (255) dengan nilai intensitas masukan (citra *grayscale* hasil operasi *contrast streching*). Dalam *matlab*, objek dikenali dengan warna yang lebih terang daripada latar belakangnya.

Oleh karena itu, diperlukan proses konversi (membalik kondisi nilai) untuk mengubah warna objek pada tulisan menjadi lebih terang dibandingkan latar belakangnya dengan menggunakan *complement.* hasil dari perintah *complement* akan mengubah warna abu-abu tua menjadi abu-abu muda dan sebaliknya, warna yang lebih muda akan menjadi lebih tua.

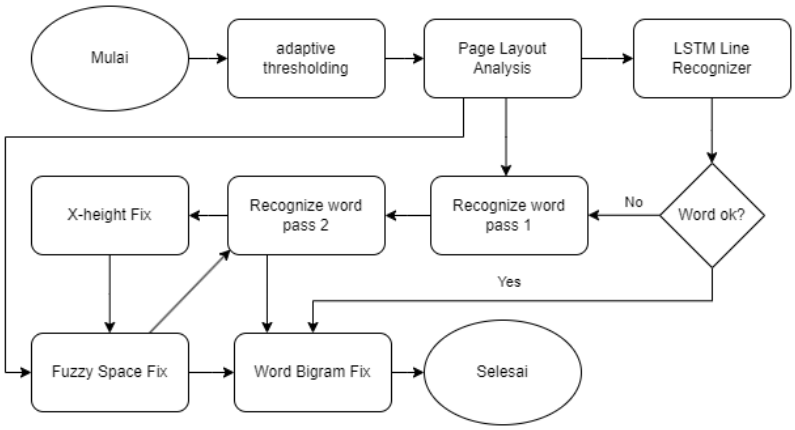
Gambar 3.6 Hasil akhir tahapan preprocessing

3.2.2 Ekstraksi Fitur

Ekstraksi fitur bertujuan untuk mengidentifikasi dan mengekstraksi atribut-atribut dari karakter yang akan digunakan untuk pengenalan. Terdapat berbagai metode yang dapat digunakan untuk mengekstraksi fitur dari citra teks, termasuk ekstraksi bentuk, tekstur, dan statistik dari karakter yang terdeteksi.

Salah satu metode ekstraksi fitur yang umum digunakan dalam OCR adalah ekstraksi bentuk dan ukuran karakter. Pada tahap ini, karakter yang telah diidentifikasi diproses untuk mengekstraksi atribut-atribut geometris seperti luas, tinggi, lebar, dan bentuk karakter. Informasi ini dapat membantu dalam membedakan karakter yang berbeda serta memperbaiki keandalan pengenalan karakter.

Untuk penggunaan OCR, *matlab* menggunakan *Tesseract engine*. Untuk penjelasan cara kerja *engine* tersebut akan dijelaskan dalam bentuk *visual* menggunakan *flowchart*.



Gambar 3.7 *Flowchart tesseract engine*

Gambar 3.7 menjelaskan tentang cara kerja *Tesseract Engine* atau *Tesseract System Architecture*. Untuk cara kerja *Engine* tersebut secara *pipeline* dimulai dari *adaptive thresholding*.

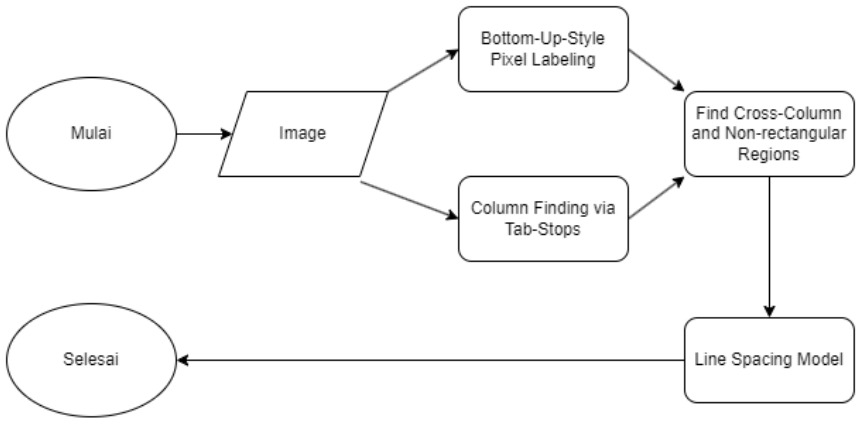
Adaptive thresholding dalam *Tesseract Engine* adalah teknik yang digunakan untuk mengubah citra menjadi citra biner sebelum proses pengenalan teks (OCR). Metode ini memungkinkan *Tesseract* untuk bekerja lebih baik dalam kondisi cahaya yang bervariasi atau ketika latar belakang citra tidak homogen. Dengan *adaptive thresholding*, citra dibagi menjadi beberapa wilayah kecil, dan *Tesseract* menghitung nilai ambang untuk setiap wilayah berdasarkan statistik lokal seperti rata-rata intensitas piksel di sekitarnya. Setiap wilayah kemudian diubah menjadi citra biner menggunakan ambang yang telah dihitung, memungkinkan *Tesseract* untuk lebih akurat memisahkan teks dari latar belakang.

Keuntungan dari *adaptive thresholding* ini terletak pada ketepatan dan pengenalan yang lebih baik oleh *Tesseract*. Dengan menggunakan ambang yang sesuai untuk setiap bagian citra, sistem dapat lebih akurat mengenali teks, terutama dalam kondisi yang sulit seperti cahaya yang bervariasi atau latar belakang yang tidak homogen. Sebagai contoh, pada sebuah halaman dokumen yang di pindai dengan pencahayaan tidak merata, *Tesseract* dapat menyesuaikan ambang secara lokal untuk membedakan bagian-bagian yang berbeda, seperti bagian atas yang lebih terang dan bagian bawah yang lebih gelap. Teknik ini memungkinkan *Tesseract* memberikan hasil OCR yang lebih andal dan akurat. setelah melakukan *adaptive thresholding* dilanjutkan dengan *Page layout analysis*.

Page layout analysis dalam *Tesseract Engine* adalah proses krusial di mana sistem mengenali dan memahami struktur halaman secara menyeluruh sebelum

melakukan OCR (*Optical Character Recognition*). Ini melibatkan identifikasi dan segmentasi elemen-elemen seperti teks, gambar, dan tabel dalam sebuah dokumen. *Tesseract* mengidentifikasi area-area di mana teks terletak, termasuk baris teks, paragraf, dan blok teks, serta gambar dan tabel yang ada. Segmentasi halaman juga dilakukan dengan membagi halaman menjadi bagian-bagian yang lebih kecil, seperti kolom atau sel, terutama saat mengenali tabel. Proses ini juga memperhitungkan *margin* dan *padding* di sekitar teks dan elemen lainnya untuk memahami batas-batas antara elemen-elemen tersebut.

Untuk penjelasannya secara visual akan dijelaskan lewat *Flowchart* di bawah ini.

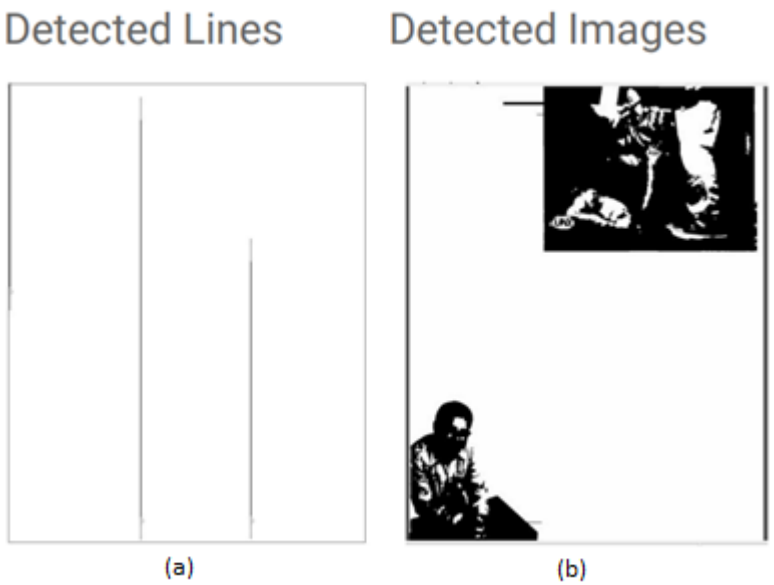


Gambar 3.8 *Flowchart* Tahapan *Page Layout Analysis*

Gambar 3.8 menjelaskan proses *page layout analysis*, dimulai dari gambar awal seperti di **Gambar 3.9**.

Gambar 3.9 *Input* gambar

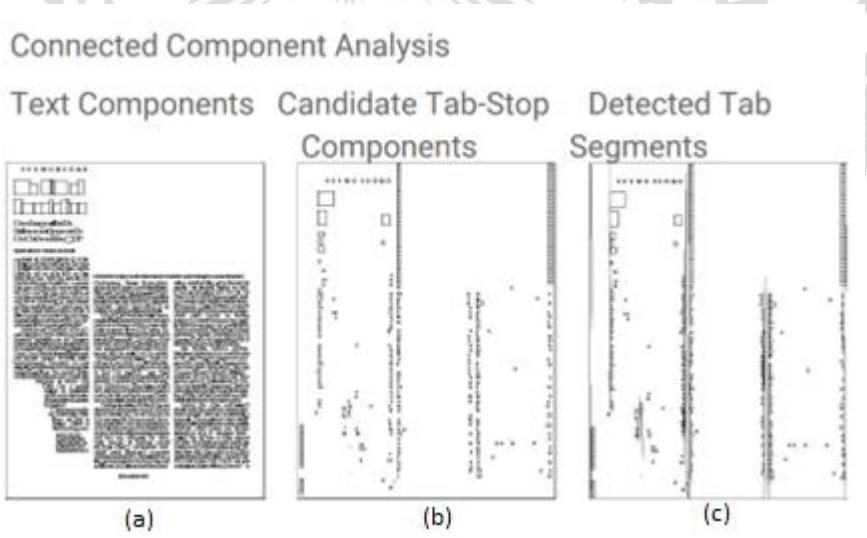
Setelah gambar tersebut dimasukkan dalam *page layout* maka akan dilakukan *detection lines* (deteksi garis) dan detected images (deteksi gambar). *Line detection* dan *image detection* adalah dua aspek krusial dalam analisis *layout* halaman (*page layout analysis*) di *Tesseract Engine*. *Line detection* digunakan untuk mengidentifikasi dan memisahkan baris-baris teks, memanfaatkan teknik seperti analisis intensitas piksel, deteksi tepi, dan transformasi Hough untuk mendeteksi garis-garis lurus. Sementara itu, *image detection* membantu *Tesseract* dalam mengenali dan memisahkan gambar atau elemen visual lainnya dari teks, menggunakan algoritma pengenalan pola dan analisis tekstur. Kedua teknik ini bekerja bersama dalam analisis *layout*, memungkinkan *Tesseract* untuk membagi dokumen menjadi bagian-bagian yang berisi teks dan gambar, sehingga memudahkan proses pengenalan teks dan mempertahankan struktur dokumen yang asli. Misalnya, dengan *line detection*, *Tesseract* mengidentifikasi baris teks, dan dengan *image detection*, ia memisahkan area-area yang berisi gambar atau ilustrasi. Integrasi deteksi ini membantu dalam segmentasi halaman yang lebih baik, menghasilkan *output* yang lebih terstruktur dan akurat saat melakukan OCR.



Gambar 3.10 Hasil Deteksi garis (a) dan gambar (b)

Setelah deteksi dilakukan untuk mencari gambar dan garis akan dilanjutkan dengan *Connected component analysis* yang memiliki beberapa fungsi lainnya seperti t*ext components, candidate tab-stop components* dan *detected tab segments*.

Text components merupakan bagian-bagian dari gambar atau dokumen yang mengandung teks, seperti baris teks, paragraf, atau blok teks, yang diidentifikasi oleh *Tesseract* untuk memahami struktur teks dalam dokumen. Identifikasi *text components*, menggunakan metode seperti *line detection*, memungkinkan *Tesseract* untuk fokus pada pengenalan teks yang sebenarnya, memisahkan teks dari gambar atau elemen visual lainnya, dan meningkatkan akurasi dalam proses OCR. Sementara itu, *candidate tab-stop components* adalah area-area dalam dokumen yang kemungkinan menjadi tempat tab-stop atau titik henti tab, digunakan untuk mengatur teks dalam kolom atau tabel. *Tesseract* mengidentifikasi *candidate tab-stop components* dengan menganalisis struktur kolom dan baris teks dalam dokumen, yang kemudian digunakan untuk menemukan posisi tab-stop yang optimal saat memproses teks. Hasil dari identifikasi ini adalah *detected tab*, yang digunakan dalam proses *layout* untuk memastikan tata letak teks yang rapi dan terstruktur, terutama dalam pengenalan tabel atau dokumen dengan format kolom yang terorganisir. Diakui bahwa *text components*, *candidate tab-stop components*, dan *detected tab* bekerja bersama dalam *connected component analysis* untuk memahami dan mengelola struktur dokumen sebelum OCR, sehingga meningkatkan akurasi dan konsistensi hasil pengenalan teks.

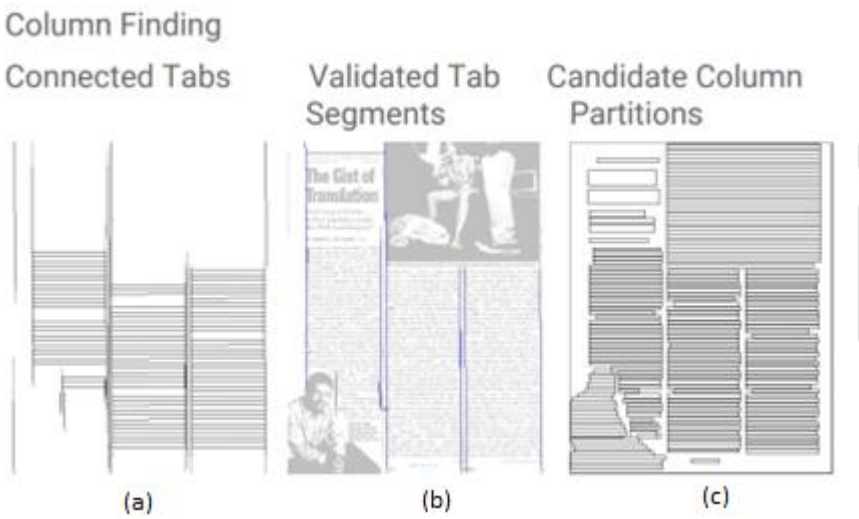


Gambar 3.11 Hasil Deteksi teks (a) dan titik henti teks (b) dan deteksi segmen (c)

Setelah dilakukan hal tersebut maka akan masuk ke bagian *column finding* dan *block finding*. Dalam analisis *layout* halaman (*page layout analysis*) di

Tesseract Engine, beberapa komponen seperti *connected tabs*, *validated tab segments*, dan *candidate column partitions* sangat berperan dalam proses penemuan kolom teks (*column finding*) dan pengaturan tata letak teks. *Connected tabs* adalah bagian dari proses *column finding* yang mencari tab-tab terhubung dengan teks dalam dokumen. Fungsi ini membantu *Tesseract* dalam menemukan posisi tab-stop yang terhubung dengan teks, yang kemudian divalidasi menjadi *validated tab segments*. Segmen tab yang divalidasi ini digunakan sebagai titik referensi dalam membagi teks menjadi kolom-kolom yang teratur, memastikan tata letak yang rapi.

Selain itu, *candidate column partitions* juga merupakan komponen kunci yang membantu *Tesseract* dalam menentukan batas-batas antar kolom-kolom berdasarkan perubahan struktur dokumen. Dengan demikian, komponen-komponen ini saling berintegrasi dalam analisis *layout* halaman, memainkan peran vital dalam proses *column finding* dan pengaturan tata letak teks untuk meningkatkan akurasi dan konsistensi dalam pengenalan teks dalam dokumen yang memiliki format kolom atau tabel yang kompleks.

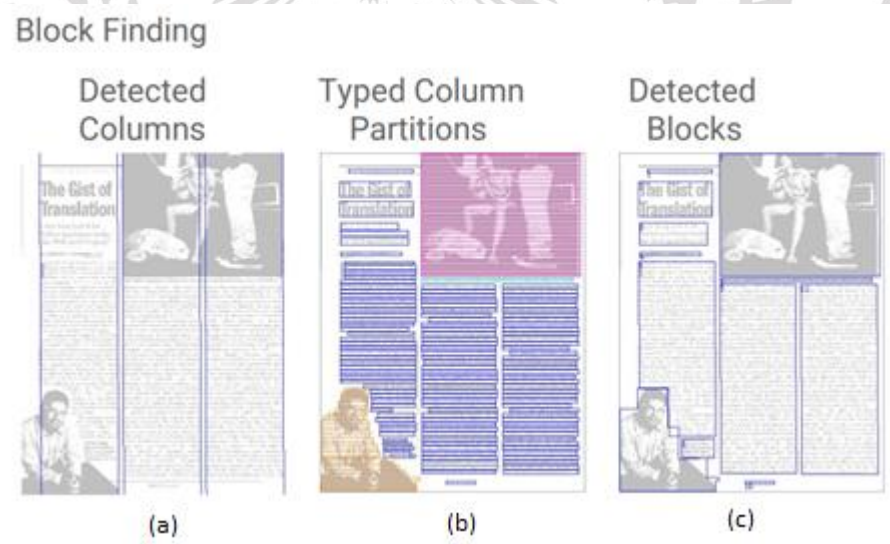


Gambar 3.12 Deteksi Batas Kalimat (a) dan validasinya (b) serta Deteksi kandidat batas antar kolom (c)

Sedangkan *block finding* adalah tahapan yang dilakukan untuk mengidentifikasi dan mengelompokkan bagian-bagian teks yang terkait dalam dokumen. *Detected columns* merupakan hasil dari proses identifikasi kolom teks oleh *Tesseract*, di mana teks dibagi ke dalam kolom-kolom berdasarkan tab-stop atau titik henti tab yang telah terdeteksi sebelumnya. Kemudian, *typed column*

partitions digunakan untuk menentukan posisi batas antara kolom-kolom teks ini dengan mempertimbangkan hubungan antara teks dan struktur dokumen. Dengan adanya *detected columns* dan *typed column partitions*, *Tesseract* kemudian menghasilkan *detected blocks* sebagai blok-blok teks yang terkait berdasarkan kolom-kolom dan partisi yang telah diklasifikasikan. Blok-blok ini dapat mengandung bagian-bagian teks dengan hubungan tematik atau struktural yang sama, memudahkan dalam memahami dan mengelola tata letak teks sebelum proses OCR.

Detected blocks kemudian menjadi titik awal dalam pengenalan teks, di mana *Tesseract* menggunakan blok-blok ini untuk mengidentifikasi dan memisahkan teks ke dalam unit-unit yang lebih kecil dan terorganisir, seperti paragraf, kalimat, atau bagian-bagian lainnya. Integrasi antara *detected columns*, *typed column partitions*, dan *detected block*s dalam *page layout analysis* sangat membantu *Tesseract* dalam mengorganisir teks secara struktural dan akurat sebelum melakukan OCR. Proses *block finding* ini meningkatkan akurasi dan konsistensi dalam pengenalan teks dalam dokumen, terutama dalam dokumen yang memiliki format kolom atau tabel yang kompleks. Dengan demikian, *block finding* dalam *Tesseract Engine* memainkan peran dalam memahami dan mengelola struktur teks, memastikan hasil OCR yang lebih baik.



Gambar 3.13 Deteksi kolum (a), kolum kalimat (b) dan kolum paragraf (c)

Setelahnya dilanjut dengan proses *find cross-column* and *non-rectangular regions*. Kedua hal tersebut merupakan bagian dari analisis *layout* halaman (*page layout analysis*). Pertama, *find cross-column regions* digunakan untuk

mengidentifikasi area dalam dokumen yang melintasi batas kolom teks, mengamati kontinuitas teks atau elemen visual yang berlanjut di sepanjang beberapa kolom. Hal ini membantu *Tesseract* dalam memahami hubungan yang berkelanjutan antar teks atau elemen visual di sepanjang dokumen, sehingga memastikan pengenalan teks dan struktur yang konsisten. Sementara itu, find non-*rectangular regions* bertujuan untuk menemukan area dengan bentuk yang tidak beraturan, tidak terbatas pada persegi atau persegi panjang seperti umumnya dokumen. Dengan menggunakan metode analisis garis tepi dan segmentasi, *Tesseract* dapat mengidentifikasi area-area ini yang sering kali mengandung informasi tambahan atau elemen visual yang tidak terikat pada struktur kolom teks.

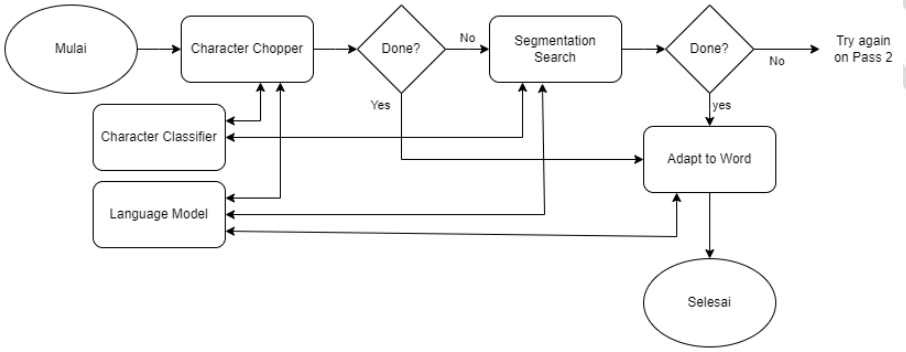
Integrasi dari kedua proses ini menjadi kunci dalam *page layout analysis*, membantu *Tesseract* dalam memahami struktur dokumen yang kompleks dan menghasilkan *output* OCR yang lebih baik. *Find cross-column regions* memungkinkan untuk mempertahankan kontinuitas teks yang melintasi batas kolom, sementara *find non-rectangular regions* membantu dalam mengelola area dengan bentuk yang tidak standar. Dengan demikian, proses ini menjadi langkah dalam mengelola tata letak halaman sebelum melakukan pengenalan teks. Hasilnya, *Tesseract* dapat mengenali dan memproses teks dengan akurat, terutama dalam dokumen dengan struktur *layout* yang kompleks atau tidak konvensional. Dengan demikian, *find cross-column* dan *non-rectangular regions* adalah bagian integral dari proses analisis *layout* halaman dalam *Tesseract Engine*. Setelahnya dilanjut dengan *line spacing*.

line spacing digunakan untuk memahami dan menyesuaikan jarak antara baris-baris teks dalam dokumen. Model ini bertujuan untuk mengidentifikasi pola jarak vertikal antara baris teks, memperbaiki segmentasi teks, dan menjaga konsistensi tata letak dokumen. Dengan menganalisis pola jarak antar baris berdasarkan garis dasar teks, *line spacing* model membantu dalam segmentasi teks yang akurat sebelum proses OCR, memastikan *output* yang konsisten dan mempertahankan struktur dokumen yang sesuai. Model ini menjadi bagian penting dalam *page layout analysis* di *Tesseract Engine*, bekerja bersama dengan komponen lain seperti *column finding* untuk menciptakan *output* OCR yang lebih baik.

Selain identifikasi dan segmentasi, *page layout* dalam *Tesseract Engine* juga melibatkan langkah-langkah seperti *deskew* (mengoreksi rotasi) gambar agar

sejajar dengan tepi halaman, serta penghapusan noise seperti bercak atau goresan yang tidak relevan pada gambar. Dengan memahami struktur halaman, *Tesseract* dapat meningkatkan akurasi pengenalan teks. Ini termasuk memisahkan teks dari gambar, mengenali data dalam tabel secara lebih baik, dan mempertahankan struktur asli dokumen seperti urutan paragraf. Dengan demikian, *page layout analysis* membantu dalam menghasilkan hasil OCR yang lebih akurat dan terorganisir, memungkinkan *Tesseract* untuk mempertahankan format asli dokumen dengan baik. Setelah *page layout analysis* dilanjutkan dengan *recognize word pass* 1 dan 2.

Recognize word pass itu memiliki beberapa fungsi di dalamnya. Fungsi tersebut akan dipaparkan dalam *flowchart* di bawah ini.



Gambar 3.14 *Flowchart recognize word pass*

Dalam **Gambar 3.14** menjelaskan diagram *pipeline* dari cara kerja *recognize word pass*, dimulai dari *character chopper* ke *segmentation search* dan selesai di *adapt word* jika tidak maka akan dilanjutkan di *word pass* ke 2. Untuk penjelasannya akan dimulai dari *character chopper*.

Character chopper dalam *Tesseract Engine* merupakan bagian dari proses *recognize word pass*, yang bertanggung jawab untuk memotong atau memisahkan karakter-karakter teks dalam kata yang diakui oleh sistem. Fungsi utamanya adalah untuk mengidentifikasi dan memisahkan karakter-karakter teks dalam sebuah kata, terutama saat karakter tersebut bersentuhan atau tumpang tindih. Proses ini penting karena dapat membantu meningkatkan akurasi pengenalan teks, terutama pada karakter yang memiliki tampilan yang kurang jelas atau bersentuhan dengan karakter lainnya. Dengan memisahkan karakter-karakter ini, *Tesseract* dapat mengenali setiap karakter dengan lebih baik dan akurat, memperbaiki hasil pengenalan kata secara keseluruhan. Dengan demikian, *character chopper*

berperan penting dalam proses *recognize word pass*, meningkatkan kualitas hasil OCR pada teks yang memiliki karakter yang bersentuhan atau tumpang tindih.

Character chopper juga menggunakan bantuan dari *character classifier* dan *language mo*del. Di dalam kedua hal tersebut juga ada beberapa fungsi lainnya, contoh dari *character classifier* ada fungsi *static* dan *adaptive* sedangkan fungsi dari *language* model ada *dictionary, char-n-grams* dan *number/punctuation parser*.

Dimulai dari *character classifier*, merupakan gabungan beberapa klasifikasi mulai dari klasifikasi untuk menemukan jarak di fitur ruang ke beberapa tujuan seperti KNN secara "*generative*" atau membagi ruang fitur ke dalam wilayah yang ditetapkan setiap kelasnya seperti SVM secara "diskriminatif". Untuk proses pengerjaannya, *Tesseract* awalnya menggunakan *skeletonization* untuk mengekstraksi fitur dari *outlines* atau garis luar tapi hasil yang diterima tidak memuaskan. Jadi *Tesseract* menggunakan cara lain yaitu *topological feature* tapi hasil yang diterima juga rapuh dengan kata lain butuh banyak variasi dari 1 huruf yang dicari untuk bisa menemukan hasil yang diinginkan. *Tesseract* juga menggunakan fitur lain yaitu *shrinking feature and inappropriate statist*ic yang menghasilkan akhir yang bagus tapi membuat data pengerjaannya terlalu kaku atau *fixed* karena hasil *statistic* jaraknya terlalu susah untuk di rubah. Tetapi dari penggunaan fitur tersebut membuat inspirasi bahwa dengan karakter yang rusak sekalipun masih bisa diidentifikasi selama fitur kata tersebut sesuai.

Akhirnya *tesseract* membuat fiturnya dengan nama 3D INT_FEATURE_STRUCT tetapi karena fitur tersebut menghasilkan hasil yang tetap panjangnya jadi tidak bisa di rubah maka *Tesseract* akhirnya membuat sebuah fitur lain dengan bantuan data secara 4 dimensi (asal kata *tesseract*) dengan fungsi INT_PROTO_STRUCT. Metode tersebut memiliki fungsi yang dinamakan fungsi jarak satu fitur ke satu *proto*. Fungsi tersebut membantu untuk menemukan jarak asli yang bisa diubah sesuai permintaan sehingga hasil yang diterima memuaskan. Dari hal tersebut menghasilkan fitur normalisasi karakter dengan satu 4-d fitur yang memiliki fungsi *Y-Position relative to baseline*, *Outline Length* (*in normalized space*), *2nd x-moment* dan *2nd y-moment*. Pengerjaan fungsi tersebut menggunakan *rating* dan *certainty* dan tidak menggunakan *probability* karena *rating* bisa menormalisasi kata atau baris jika mau dan transkripsi dengan Panjang yang berbeda cukup sebanding sedangkan *certainty* untuk mengukur klasifikasi

absolut mutlak dan pengganti probabilitas log dan berguna untuk memutuskan apa yang memerlukan pekerjaan lebih banyak. Juga untuk membandingkan produk dengan probabilitas memerlukan banyak perhitungan yang tidak ketat.

Untuk normalisasi ada banyak tahapan yang dilakukan di *Tesseract* dengan aturan :

1. Untuk gambar hanya API yang tahu *scaling* dan *cropping* dari asal gambar dan *tesseract* tidak akan menyentuhnya.
2. Untuk *C_BLOBs* dibatasi hanya berorientasi piksel, tidak bisa diskala dan diputar selain oleh kelipatan 90 derajat, satu-satunya perbedaan dengan gambar adalah kemungkinan rotasi blok untuk menjaga garis teks tetap horizontal.
3. Untuk *TBLOBs* adalah sebagai *input* untuk pemotong dan klasifikasi, dengan tambahan rotasi untuk klasifikasi.

Untuk normalisasi klasifikasi berdasarkan kata menggunakan TBLOB, penggolong kemudian melakukan normalisasi lebih lanjut untuk ekstraksi fitur menggunakan normalisasi garis datar, normalisasi karakter dan normalisasi non-linear. Untuk pengaplikasi normalisasi ada perbedaan untuk penggunaannya antara lain untuk *baseline/x-height* digunakan untuk penggolong yang *adaptive*, menggunakan *centroid* x dan y pada garis dasar, mengubah skala secara seragam oleh tinggi-x, melihat sup/super sebagai kelas yang berbeda dan baik dalam mengabaikan kebisingan (*noise*).untuk *character moments* digunakan untuk penggolong statis, diatur pada *centroid*, mengubah skala oleh momen kedua secara *independent* dalam x dan y, menghilangkan banyak variasi *font*, membuat beberapa bentuk ambigu, membuat *sub/superscript* muncul normal dan mudah tertipu oleh kebisingan (*noise*).

Selain *character classifier* juga ada *language model*, merupakan komponen kunci yang berperan dalam meningkatkan akurasi pengenalan teks. Salah satu fungsi utamanya adalah menggunakan *dictionary* untuk memeriksa kata-kata yang diakui oleh *Tesseract*. Jika sebuah kata tidak terdapat dalam *dictionary*, kemungkinan besar akan dianggap sebagai kesalahan pengenalan yang perlu diperbaiki. Selain itu, *char-n-grams*, yang menganalisis urutan karakter dalam kata-kata yang diakui, membantu *Tesseract* dalam mempertimbangkan kombinasi karakter berurutan yang umum dalam bahasa tertentu. Fungsi lainnya adalah *number*/*punctuation* parser, yang bertanggung jawab untuk mengenali dan

memproses angka serta tanda baca dalam teks. Hal ini memastikan bahwa *Tesseract* dapat membedakan antara angka dan kata, serta memperlakukan tanda baca dengan benar dalam proses pengenalan teks.

Semua fungsi ini bekerja bersama dalam proses *recognize word pass* di *Tesseract Engine*, di mana *dictionary* memeriksa kata-kata, *char-n-grams* menganalisis urutan karakter, dan *number/punctuation* parser memproses angka dan tanda baca. Integrasi dari semua ini memungkinkan *Tesseract* untuk mempertimbangkan konteks bahasa yang lebih luas dan meningkatkan akurasi pengenalan teks secara keseluruhan. Dengan demikian, *language model* dalam *Tesseract Engine* menjadi kunci dalam meningkatkan kualitas hasil OCR, dengan memperhitungkan aturan bahasa, kombinasi karakter, serta angka dan tanda baca dalam teks yang diakui.

Setelah proses selesai maka kata yang didapat akan langsung diadaptasi ke kata dengan pencarian langsung dari *language mo*del yang telah disediakan. Untuk proses fungsi *character chopper* tidak selesai dilanjut dengan proses *segmentation search*. *Segmentation search* di *tesseract* hanya memisahkan dan menggabungkan kata yang dibutuhkan. Dengan kata lain mengurangi *over-segmentation* sehingga memaksimalkan tahapan potongan untuk mengurangi pemanggilan ulang dengan pengorbanan pada akurasi. Setelah segmentasi maka kata yang selesai akan diadaptasi ke kata jika tidak maka akan dilanjut ke *pass* 2 dan selesai *pass* 2 akan masuk ke *x-height fix*.

x-height fix merupakan proses yang bertujuan untuk mengatasi masalah pengenalan teks yang terkait dengan tinggi huruf (*x-height*) yang bervariasi. *X-height* adalah tinggi huruf bagian tengah, seperti huruf "x" atau "o", dan perbedaan dalam *x-height* dapat memengaruhi pengenalan teks. Proses *x-height fix* bekerja dengan cara menyesuaikan tinggi huruf dalam proses pengenalan teks, terutama untuk huruf-huruf yang memiliki *x-height* yang tidak standar. Dengan memperbaiki tinggi huruf ini, *Tesseract* dapat meningkatkan akurasi pengenalan teks, terutama pada dokumen yang memiliki variasi tinggi huruf yang signifikan.

Dalam proses *x-height fix*, *Tesseract Engine* menggunakan informasi tentang *font* dan konteks dokumen untuk menyesuaikan tinggi huruf secara dinamis. Misalnya, jika ada variasi *x-height* yang tidak biasa dalam dokumen, seperti huruf "o" yang lebih tinggi dari biasanya, *x-height fix* akan menyesuaikan tinggi huruf ini agar lebih sesuai dengan yang diharapkan. Hal ini membantu dalam meningkatkan

konsistensi dalam pengenalan teks dan mengurangi kesalahan yang disebabkan oleh variasi tinggi huruf yang tidak standar. Dengan demikian, *x-height fix* adalah langkah penting dalam proses pengenalan teks di *Tesseract Engine*, membantu meningkatkan akurasi dan konsistensi hasil OCR, terutama dalam dokumen dengan *font* atau huruf yang bervariasi.

Setelah proses *x-height fix* dilanjut oleh proses *Fuzzy space fix*. proses tersebut digunakan untuk mengatasi masalah spasi yang tidak jelas atau tidak konsisten antara kata-kata dalam teks yang diakui. Ketika teks diakui, terkadang terjadi kesalahan dalam penentuan spasi antara kata-kata, terutama saat ada teks yang bergabung atau tumpang tindih. Proses ini menggunakan pendekatan "*fuzzy*", yang berarti mendekati atau kurang pasti, untuk menentukan spasi yang seharusnya antara kata-kata. *Tesseract Engin*e akan memeriksa konteks teks dan karakter di sekitar spasi yang ambigu, lalu membuat perkiraan atau estimasi untuk menambahkan atau mengurangi spasi yang diperlukan. Dengan melakukan ini, *fuzzy space fix* membantu meningkatkan konsistensi dan kejelasan teks yang dihasilkan setelah proses OCR.

Proses ini berkontribusi dalam memisahkan kata-kata yang seharusnya terpisah namun terlihat bergabung, serta mengurangi kemungkinan terjadinya kesalahan pengenalan teks akibat spasi yang salah. Secara keseluruhan, *fuzzy space fix* adalah langkah penting dalam *post-processing* teks di *Tesseract Engine*. Proses ini membantu mengoreksi spasi yang ambigu antara kata-kata, meningkatkan kejelasan dan konsistensi teks yang dihasilkan setelah OCR. Dengan demikian, *fuzzy space fix* berkontribusi dalam meningkatkan akurasi hasil pengenalan teks dalam *Tesseract Engine*, terutama dalam dokumen yang memiliki spasi yang tidak jelas atau tidak konsisten.

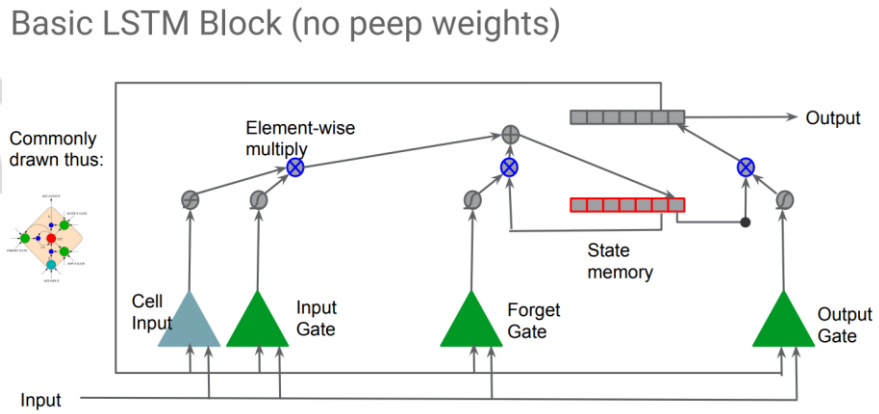
Proses yang terakhir sebelum selesai adalah *word bigram fix*. proses tersebut digunakan untuk memperbaiki kesalahan pengenalan teks yang terjadi pada kata-kata yang berdekatan, atau yang disebut dengan *bigram*. *Bigram* adalah dua kata yang berurutan dalam teks, dan *word bigram fix* bertujuan untuk mengoreksi kesalahan pengenalan pada pasangan kata-kata ini. Dalam beberapa kasus, *Tesseract* dapat salah mengenali *bigram* karena kemungkinan variasi karakter atau kata yang mirip.

Proses ini bekerja dengan memeriksa konteks dan makna dari *bigram* yang terdeteksi, kemudian mencocokkan dengan kata-kata yang mungkin benar

berdasarkan *dictionary* atau model bahasa. Jika terdapat *bigram* yang tidak sesuai atau tidak cocok dengan kata-kata yang umumnya muncul berdampingan, *word bigram fix* akan mencoba untuk mengoreksi dan menyesuaikan pasangan kata tersebut. Hal ini membantu meningkatkan akurasi pengenalan teks, terutama dalam kasus di mana ada variasi dalam urutan kata atau kesalahan pengenalan pada pasangan kata yang sering muncul bersama.

Secara keseluruhan, *word bigram fix* merupakan salah satu langkah dalam *post-processing* teks di *Tesseract Engine* yang berfokus pada memperbaiki kesalahan pengenalan pada pasangan kata-kata yang berdekatan. Proses ini membantu untuk memastikan keakuratan teks yang dihasilkan setelah proses OCR, dengan mencocokkan dan memperbaiki *bigram* yang tidak sesuai dengan kata-kata yang umumnya muncul berdampingan. Dengan demikian, *word bigram fix* berperan penting dalam meningkatkan kualitas dan keakuratan hasil pengenalan teks dalam *Tesseract Engine*.

Untuk LSTM *line recognizer* baru ditambahkan pada beberapa tahun ke belakang, LSTM digunakan karena mempunyai hasil yang sama dengan HMM dan DNN serta lebih baik dari metode lainnya. Untuk integrasi LSTM dengan *tesseract* mulai dari kode LSTM didasari oleh implementasi *OCROpus python*, memperluas kemampuan termasuk 2-d dan ukuran input bervariasi, sepenuhnya berintegrasi dengan *tesseract* pada Tingkat kelompok kata yang mirip, visualisasi dengan API *viewer* yang ada dan lainnya. Hasil integrasi tersebut membuat sebuah jaringan *tesseract* dengan *string* yang ringkas, kemampuan terbatas tapi sangat fleksibel dalam Batasan tertentu dan mudah digunakan serta sedikit yang perlu di pelajari.



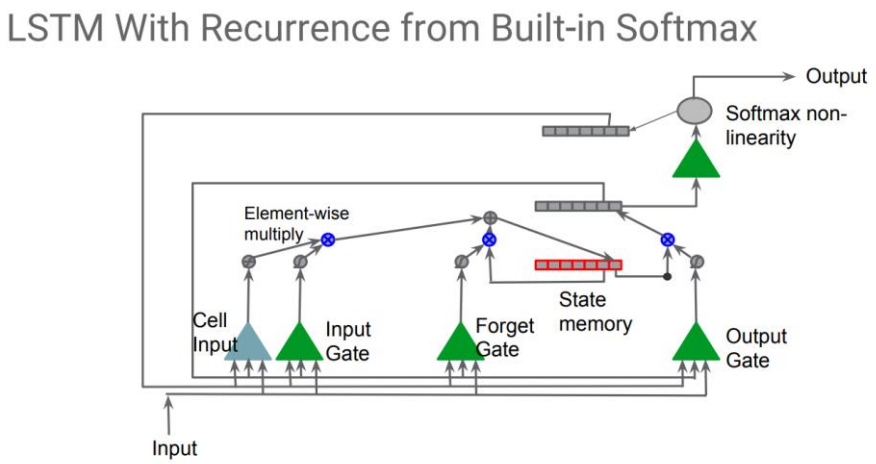
Gambar 3.15 Sistem LSTM

Gambar 3.15 menjelaskan cara kerja dasar LSTM, biasanya sering digunakan dalam *recurrant neural networks* (RNNs). RNN merupakan jenis jaringan saraf tiruan yang mampu memproses data berurutan, seperti teks atau data deret waktu. LSTM adalah jenis khusus dari RNN yang mampu belajar dependensi jangka panjang dalam data.

Berikut adalah penjelasan tentang bagian LSTM dari **Gambar 3.15**:

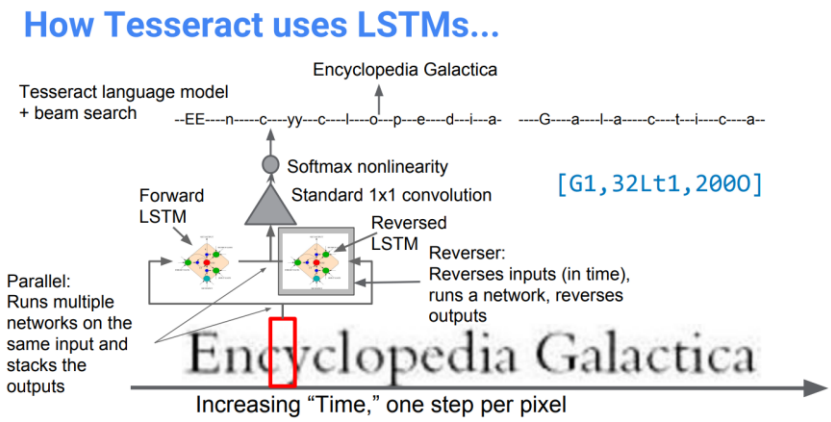
1. *Cell input* : inti blok LSTM di mana komputasi sebenarnya berlangsung dan Berisi memori untuk penyimpanan informasi.
2. *Input gate* : Gerbang masukan mengontrol bagian mana dari data masukan baru yang diizinkan untuk memperbarui memori sel.
3. *Forget gate* : Gerbang lupa mengontrol bagian mana dari memori sel yang dilupakan atau dipertahankan.
4. *Output gate* : Gerbang keluaran mengontrol bagian mana dari memori sel yang digunakan untuk menghasilkan *output* blok LSTM.
5. *Element-wise multiply* : operasi matematika yang digunakan untuk mengendalikan aliran informasi dalam LSTM.
6. *State memory* : memori jangka panjang yang digunakan untuk menyimpan informasi yang relevan dari masa lalu.

Cara kerja LSTM dimulai Ketika data masuk ke dalam *input* dan menerima *output* dari Langkah sebelumnya. Setelahnya data akan dihitung dengan *element-wise multiply* berdasarkan *vector* atau matriks yang sama dari *cell input* dan *input gate*. Setelah data diolah, *forget gate* akan menentukan seberapa banyak informasi yang akan dilupakan dari *state memory*. data dari *forget gate* dan *state memory* diolah melewati *element-wise multiply* yang kedua. Olahan dari kedua data *element-wise multiply* akan digabung dan masuk ke *memory state*, Di mana *memory state* akan mempertimbangkan informasi lama yang dipertahankan dan informasi baru yang dimasukkan. Setelahnya data akan menuju *element-wise multiply* yang terakhir Bersama dengan data dari *output gate*, Di mana *output gate* menentukan banyaknya informasi yang keluar dari *state memory* untuk menjadi *output* aktual dari LSTM. Tetapi LSTM juga memiliki tambahan layer *classification* Bernama *softmax* seperti gambar di bawah ini.



Gambar 3.16 Sistem LSTM dengan *softmax*

Berdasarkan **Gambar 3.16**, menambahkan *softmax* ke dalam LSTM yang digunakan karena memberikan distribusi probabilitas terhadap kelas karakter yang mungkin, memungkinkan klasifikasi *multi-class*. Ini memungkinkan untuk membuat prediksi dengan keyakinan yang terkait, membantu dalam estimasi kesalahan dan pelatihan model. Penggunaan *softmax* memastikan bahwa sistem dapat mengurusi tugas pengenalan karakter dengan efektif dan memberikan hasil yang dapat diandalkan.



Gambar 3.17 sistem LSTM *Tesseract*

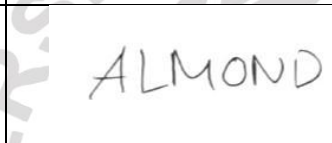
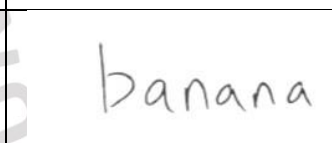
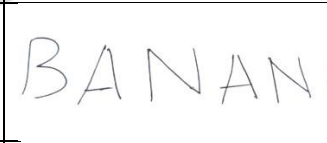
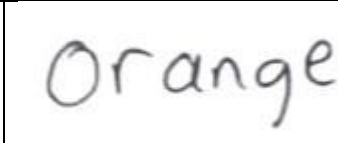
Gambar 3.17 menjelaskan cara *tesseract* menggunakan LSTM, dengan cara menjalankan kedua LSTM secara *parallel* dengan *forward* LSTM dan *reversed* LSTM di mana hasil kedua LSTM akan menuju ke *standard* 1x1 *convolution* untuk mengurangi dimensi atau kedalaman data. Hal tersebut membantu mengurangi beban komputasi dan parameter modal tanpa mengorbankan kapasitas representasi.

Dengan kata lain dapat membantu mengoptimalkan performa model dengan mengurangi kompleksitasnya.

Setelahnya akan menuju *softmax nonlinearity*. Hal tersebut menghasilkan distribusi probabilitas untuk setiap kelas karakter, dan karakter dengan probabilitas tertinggi dipilih sebagai *output* prediksi untuk wilayah masukan. Ketika proses ini selesai maka akan langsung menuju *word bigram fix* jika tidak maka akan masuk ke *recognize word pass*. Berikut ini adalah hasil tes beberapa data awal yang dihasilkan dari proses ini.

Untuk *output* yang dihasilkan adalah dalam bentuk tulisan digital yang didapat dari gambar tulisan yang dimasukkan. Contoh hasil dari ekstraksi fitur menggunakan data *dummy* berada di bawah ini.

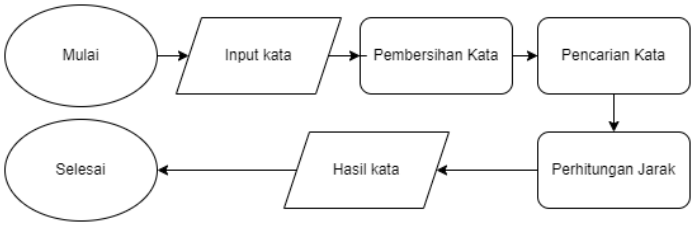
Tabel 3.1 Hasil Ekstraksi Fitur (OCR)

No	Gambar awal	Gambar akhir	Hasil teks
1			ALMOND
2			APFKE
3			9flflAna
4			BAN/Awx
5			CHEPRY
6			LEMON
7			Ordnafl
8			OPANQE

Sumber: Data Sendiri (Rakhmadhan Rizky)

3.2.3 Persamaan Kata

Pada hasil yang telah didapat setelah melakukan proses Ekstraksi Fitur (OCR) menggunakan *Tesseract*, dengan menggunakan *matlab*, akan dihasilkan hasil deteksi citra berupa teks. teks tersebut akan dijadikan masukan untuk mendapatkan kata yang lebih tepat. kata masukan tersebut akan diproses melalui beberapa tahapan terlebih dahulu sebelum masuk ke proses persamaan kata menggunakan metode *Hamming distance*. *Flowchart* di bawah ini akan menjelaskan tahapan yang terjadi.



Gambar 3.18 *Flowchart* Tahapan Persamaan kata

Gambar 3.18 menjelaskan mengenai tahapan persamaan kata dalam penelitian. Adapun penjelasan mengenai tiap tahapan persamaan kata sebagai berikut:

1. Tahap awal dimulai dengan membersihkan *input* kata yang didapat dari ekstraksi fitur. *Input* kata dibersihkan dari simbol hingga angka yang terbawa lewat ekstraksi dan diganti dengan spasi untuk mempermudah perhitungan jarak.
2. Setelah proses pembersihan kata selesai, dilanjut dengan proses pencarian kata. *Input* kata dimasukkan dalam algoritma untuk mencari kata yang sesuai, untuk penelitian ini *input* kata yang dipakai adalah buah karena tidak berubah-ubah. ada 3 tahapan dalam proses ini yang pertama dengan cara mencocokkan kata awal dengan kata yang ada di basis data, yang kedua dengan cara mencocokkan kata awal yang terpotong secara utuh dan tidak terpisah dengan basis data dan yang ketiga dengan cara mencocokkan kata yang dipisah per huruf dan sesuai posisi huruf dengan *database*.
3. Setelah melakukan pencarian kata dilanjut dengan perhitungan jarak menggunakan metode *hamming distance*. Cara kerja metode *hamming distance*

yaitu memecah kata awal dan kata yang ditemukan sesuai menjadi huruf satu persatu untuk dibandingkan katanya per huruf dan sesuai posisi. Jika kata yang ditemukan memiliki beberapa perbedaan huruf dari kata awal maka total huruf dari kata awal akan dikurangi dengan total huruf dari kata yang sesuai dengan syarat posisi huruf sama dan diberi nilai persentase untuk membantu penilaian.

4. Langkah selanjutnya adalah menghitung akurasi prediksi kata yang ditemukan dengan persamaan akurasi di bawah ini:

$$Akurasi = \frac{Kata\ benar}{Total\ Kata} \times 100\% \quad (3.1)$$

Berikut ini adalah hasil yang telah melewati proses 1 hingga 3 persamaan kata.

Tabel 3.2 Hasil Proses 1, 2 dan 3 Persamaan Kata

No.	*Input* Kata	Hasil Kata	Tingkat Kemiripan	Jarak *Hamming*
1	ALMOND	Almond	100.0	0
2	APFKE	Apple	60.0	2
3	9flflAna	Papaya	12.5	7
4	BAN/Awx	Banana	42.857142857142854	4
5	CHEPRY	Cherry	83.33333333333334	1
6	LEMON	Lemon	100.0	0
7	Ordnafl	Orange	42.857142857142854	4
8	OPANQE	Orange	66.66666666666666	2

Sumber: Data sendiri (Rakhmadhan Rizky)

Setelah Proses 1 hingga 3 selesai, dilanjut pencarian nilai kebenaran dari hasil kata dengan *ground truth* di bawah ini.

Tabel 3.3 Hasil Pencocokan *Ground Truth*

No.	Hasil Kata	Ground Truth	Kecocokan
1	Almond	Almond	Cocok
2	Apple	Apple	Cocok
3	Papaya	Banana	Tidak cocok
4	Banana	Banana	Cocok
5	Cherry	Cherry	Cocok

6	Lemon	Lemon	Cocok
7	Orange	Orange	Cocok
8	Orange	Orange	Cocok

Sumber: Data sendiri (rakhmadhan rizky)

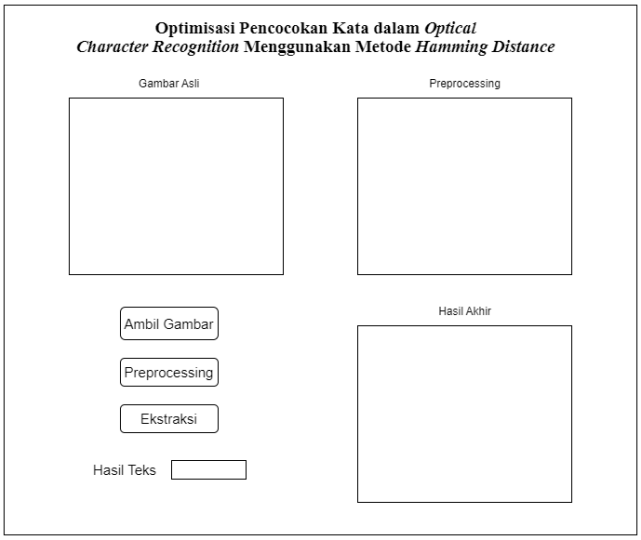
Pengerjaan proses ini dilakukan menggunakan *python* di *google colab*. Dikarenakan jika proses ini dilakukan di *matlab* ada beberapa kendala yang ditemui dari sedikitnya referensi hingga kurang terbaru beberapa referensi tentang *hamming distance* pada *matlab*. Sedangkan untuk *hamming distance* pada *python* terus berkembang hingga ada beberapa *library* khusus untuk *hamming distance* dan juga banyak referensi terbaru. Untuk pemakaian *google colab* dikarenakan *website* tersebut gratis dan juga memiliki beberapa kelebihan daripada menggunakan aplikasi lokal juga rata-rata lebih baik daripada perangkat pribadi. mulai dari *tools* dan *library* yang sudah tersedia, kemampuan *hardware* yang lebih mumpuni dari rata-rata laptop/komputer yang ada dan sifatnya yang tersimpan secara *online* bisa lebih mudah diakses di mana saja.

3.3 Kebutuhan Perangkat Keras dan Perangkat Lunak

Pada penelitian ini alat yang digunakan berupa *software* dan *hardware*. Alat - alat yang digunakan antara lain untuk *hardware* ada laptop dengan spesifikasi *processor* Amd 5 4600H, Ram 16GB dan *Video Graphics Array* (VGA) *Nvidia Geforce* GTX 1650 Ti serta *Operating system* Windows 10 home 64-bit. *Software* yang digunakan adalah *Matlab* R2021a dan *Google colab* versi terbaru.

3.4 Perancangan Antarmuka Sistem

Pengembangan Optimisasi Pencocokan Kata dalam *Optical Character Recognition* Menggunakan Metode *Hamming Distance* menggunakan sistem berbasis *Graphical User Interface* (GUI) menggunakan *matlab* dan *google colab*. Tampilan rancangan GUI matlab seperti pada **Gambar 3.19**.



Gambar 3.19 Rancangan GUI *matlab*

Tahapan untuk menggunakan sistem GUI tersebut dimulai dengan mengambil gambar dengan ekstensi .JPG menggunakan tombol "Ambil Gambar", setelah memilih gambar yang akan digunakan maka gambar akan muncul pada bagian "Gambar Asli". Selanjutnya dengan menekan tombol "*Preprocessing*" untuk melakukan *preprocessing* pada gambar yang dipilih dan akan keluar hasilnya pada bagian "*Preprocessing*". Langkah terakhir dengan menekan tombol "Ekstraksi" untuk melakukan ekstraksi menggunakan OCR dan hasil gambar akan ditunjukkan pada bagian "Hasil Akhir" dan hasil teks akan ditunjukkan pada bagian "Hasil teks".

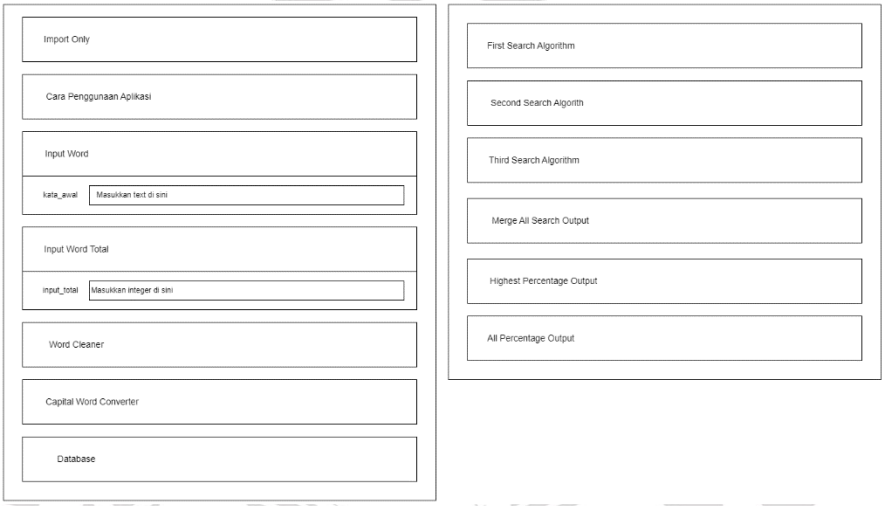
Tabel 3.4 Rancangan Menu Sistem Pencocokan Kata Tulisan Tangan

No.	Nama	Jenis	Keterangan
1	Ambil Gambar	Push Button	Membuka file gambar tulisan tangan dan menampilkan gambar asli
2	Preprocessing	Push Button	Melakukan preprocessing dan menampilkan gambar hasil preprocessing
3	Ekstraksi	Push Button	Melakukan Ekstraksi Fitur dan menampilkan gambar hasil Ekstraksi Fitur
4	Hasil Teks	Edit Text	Menampilkan Hasil Teks Ekstraksi Fitur
5	Gambar asli	Axes	Menampilkan Gambar Asli

6	Preprocessing	Axes	Menampilkan Gambar Hasil preprocessing
7	Hasil Akhir	Axes	Menampilkan Gambar Hasil Ekstraksi Fitur

Sumber: Sistem sendiri (rakhmadhan rizky)

Untuk tampilan rancangan GUI *google colab* seperti pada **Gambar 3.20**.



Gambar 3.20 Rancangan GUI *Google colab*

Tahapan untuk menggunakan sistem GUI tersebut dimulai dengan menghubungkan *runtime* yang telah di *hosting google*, setelahnya mengisi bagian "kata_awal" dengan kata yang dicari atau yang didapat dari hasil OCR sebelumnya. Selanjutnya mengisi bagian "input_total" dengan total kata yang dicari atau yang didapat dari hasil OCR sebelumnya. Setelah pengisian kedua bagian selesai dilanjut dengan menjalankan semua kode yang ada dengan cara mengeklik *runtime* dan mengeklik jalankan semua atau dengan cara menekan dan menahan tombol *control* (CTRL) serta menekan tombol F9. Setelah semua kode berjalan hingga menunjukkan tanda centang hijau di sebelah kiri kode atau tulisan selesai pada bagian bawah *google colab*, hasil bisa dilihat di bagian *file* (UI sebelah kiri urutan ke 5 dari atas).

Untuk hasil dibagi menjadi 2 yaitu "hasil_persamaan" dan "hasil_persamaan_tertinggi" untuk memudahkan penggunaan selanjutnya yaitu pengecekan akurasi. Untuk melakukan penyimpanan secara *offline* bisa dilakukan pengunduhan dengan cara klik kanan pada *file* yang diinginkan dan klik *download* atau unduh.

3.5 Skenario Pengujian

Pengujian pada penelitian ini dilakukan untuk model yang dihasilkan dari ekstraksi fitur OCR dan persamaan kata *Hamming Distance* sebelumnya. Pada penelitian ini model yang dibangun akan dilihat keakuratannya dengan memperhatikan parameter pengujian. Pada metode ekstraksi fitur OCR yang akan diperhatikan adalah kecocokan kata dengan hasil sedangkan untuk metode persamaan kata akan memperhatikan perbedaan kata dengan kata awal. berikut merupakan skenario pengujian.

1. Pengujian sistem menggunakan *blackbox* untuk menguji apakah sistem sudah siap digunakan juga mengetes sekuritas sistem.
2. Pengujian akurasi yang didasari oleh parameter pada metode yang digunakan yaitu OCR, parameter yang digunakan dalam pengujian ini seberapa dekatnya hasil dengan *ground truth*. Di mana semakin jauh hasil dengan *ground truth* maka hasil akurasi yang dihasilkan semakin buruk sedangkan jika semakin dekat dengan *ground truth* maka akurasi yang dihasilkan juga semakin baik. Minimal akurasi yang dicari adalah 50%.
3. Pengujian akurasi yang didasari oleh parameter dengan metode yang diujikan yaitu dengan menggunakan jarak kata dengan perbedaan kata dari maksimum dari total huruf yang dicari dikurangi dengan satu hingga tiga huruf, dengan kata lain jika huruf awal totalnya adalah 6 seperti pisang maka yang perbedaan huruf yang digunakan adalah 3 sampai 5 huruf yang berbeda dari kata yang dicari.
4. Pada proses pengujian yang dilakukan akan menggunakan metode uji akurasi yang diambil dari perbandingan nilai yang didapat dengan kata benar yang sesuai dengan *ground truth* yang ada di bandingkan dengan total kata yang diuji. Nilai hitung akurasi dapat dilakukan dengan persamaan 3.1 menggunakan data yang ada pada **Tabel 3.3**.

$$Akurasi = \frac{7}{8} \times 100\% = 87{,}5\ \%$$