JTT (Jurnal Teknologi Terapan) | Volume 10, Nomor 1, Maret 2024 p-ISSN 2477-3506
e-ISSN 2549-1938

ANALISIS PENGGUNAAN KOMBINASI *WORD EMBEDDING*, *RESNET*, DAN *GRU* PADA MODEL PIX2CODE

Fadly Triansyah Rahman[1], Farhan Rahmat Abdillah[2] , Transmissia Semiawan[3], Nurjannah Syakrani [4]

[1,2,3,4]Teknik Komputer dan Informatika, Politeknik Negeri Bandung
Email: [1]fadly.triansyah.tif417@polban.ac.id, [2]farhan.rahmat.tif417@polban.ac.id,
[3]transmissia@jtk.polban.ac.id, [4]nurjannahsy@jtk.polban.ac.id

Abstrak

Model pix2code merupakan model *machine learning* yang dapat mengotomasi proses implementasi *mockup* menjadi *code* dari suatu *Graphical User Interface* (GUI). Dikembangkan pertama kali oleh Tonny Beltramelli, model ini memiliki nilai akurasi sejumlah 77%. Pengembangan dari model pix2code dinilai penting karena dengan meningkatkan akurasi, maka hasil *code* yang dihasilkan oleh model semakin akurat. Penelitian ini mengembangkan model pix2code dengan penggunaan kombinasi metode *word embedding*, *Residual Network* (ResNet), dan GRU. Data yang digunakan berasal dari penelitian Tony Beltramelli sebanyak 3500 *dataset* yang terdiri dari *image mockup* dan *context* dari *image* tersebut. Hasil penelitian ini menunjukkan bahwa kombinasi *word embedding*, ResNet dan GRU telah berhasil meningkatkan akurasi model pix2code dari 77% menjadi sebesar 80%. Selain itu diperoleh nilai variansi distribusi nilai akurasi model pix2code menurun dari 0.078782 menjadi 0.046656. Hal ini menunjukkan bahwa dengan variasi data yang digunakan, akurasi dari model pix2code yang dikembangkan menjadi lebih stabil.

Kata Kunci: Otomasi *Code*, *Deep Learning*, *Word embedding*, *Residual Network*, *GRU*

Abstract

The pix2code model is a machine learning model that can automate the mockup implementation process into code from a Graphical User Interface (GUI). First developed by Tony Beltramelli, this model has a rating of 77%. The development of the pix2code model is considered important because, by increasing accuracy, the code generated by the model is accurate. This study develops a pix2code model using a combination of word embedding, Residual Network (ResNet), and GRU methods. The data used comes from the research of Tony Beltramelli as many as 3500 datasets consisting of mockup images and the context of the image. The results of this study indicate that the combination of word embedding, ResNet, and GRU has succeeded in increasing the accuracy of the pix2code model from 77% to 80%. In addition, the value of the distribution variance of the pix2code model's accuracy decreased from 0.078782 to 0.046656. This shows that with the variation of the data used, the accuracy of the developed pix2code model becomes more stable.

Keywords: *Code Automation, Deep learning, Word embedding, Residual Network, GRU*

I. PENDAHULUAN

Proses implementasi *mockup* dalam industri *software* merupakan proses yang dilakukan berulang-ulang untuk mengimplementasikan hal yang sama. Sebagai contoh *mockups* dari suatu aplikasi *website* dapat terdiri dari beberapa halaman dengan struktur dan penempatan elemen dari setiap halaman.

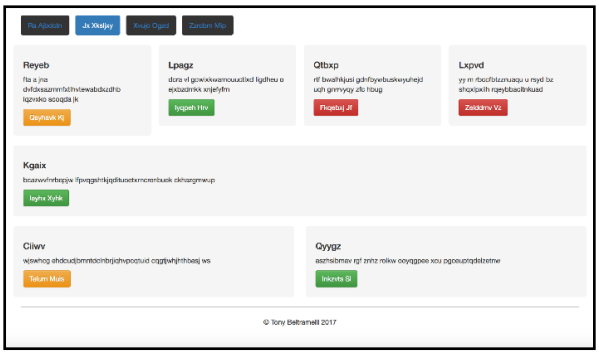
Karena permasalahan tersebut, Tony Beltramelli (2017) berhasil membangun suatu model *machine learning* bernama pix2code. Model ini mengotomatisasi proses implementasi *Graphical User Interface* (GUI) pada tiga buah platform yakni *web*, android, dan ios dengan *input* berupa gambar *mockup* saja. Model pix2code bekerja berdasarkan tiga buah model yang berbeda. Model pertama yaitu *vision model* dibangun dengan menggunakan

Convolutional Neural Network (CNN), model kedua dan ketiga yaitu *languange model* dan d*ecoder model* dibangun dengan menggunakan *Long short-term memory* (LSTM) (Beltramelli, 2017).

Vision model berfungsi untuk melakukan *encoding input* dari *image mockups* yang diberikan. *Languange model* digunakan untuk melakukan *encoding* terhadap *context* dari *input image* tersebut. *Context image* diproses oleh *language model* mengandung *Domain Specific Language* (DSL) untuk merepresentasikan hal-hal yang ada pada *mockup* dengan token-token tertentu. *Decoder model* berfungsi untuk melakukan *decode* dari gabungan *output vision* dan *language model* (Beltramelli, 2017).

Model pix2code menggunakan *one-hot encoding* dalam merepresentasikan kata pada *language model*. *One-hot encoding* adalah suatu teknik yang digunakan dalam *deep learning* untuk dapat mengubah suatu data menjadi representasi *one-hot vector*. Representasi *one-hot* akan berisi sekumpulan bit di mana satu kolom dalam data akan memiliki satu bit bernilai satu sedangkan bit lainnya bernilai nol, representasi ini juga dikenal dengan vector renggang/*sparse vector* (Patterson & Gibson, 2017).

Gambar 1 menunjukkan contoh data *mockup*, dan Gambar 2 menunjukkan contoh data *context image* yang diadopsi dari penelitian (Beltramelli, 2017) untuk digunakan dalam penelitian ini.



Gambar. 1 Contoh *mockup* dari (Beltramelli, 2017)

```
header {
 btn-inactive, btn-active, btn-inactive, btn-inactive
}
row {
 quadruple {
  small-title, text, btn-orange
 }
 quadruple {
  small-title, text, btn-green
 }
 quadruple {
  small-title, text, btn-red
 }
 quadruple {
  small-title, text, btn-red
 }
}
row {
 single {
  small-title, text, btn-green
 }
}
row {
 double {
  small-title, text, btn-orange
 }
 double {
  small-title, text, btn-green
 }
}
```

Gambar. 2 Contoh *context image* dari (Beltramelli, 2017), yang berisikan DSL representasi *mockup*

Domain penelitian untuk mengotomatisasi pembuatan *code* GUI dengan menggunakan *input* berupa gambar *mockup* dapat dikatakan baru. Matej Balog dkk. (2017), berhasil membangun suatu sistem yang dapat menyelesaikan beberapa permasalahan dalam *online programming challenges* di dunia nyata dengan menyusun solusi yang berupa *code* program untuk *challenges* tersebut (Balog et al., 2017).

Berdasarkan penelitian (Beltramelli, 2017), ditemukan bahwa token DSL baru yang dihasilkan oleh pix2code memiliki akurasi senilai 77%. Akurasi dari model pix2code akan berpengaruh terhadap *code* yang hasilkan oleh model tersebut. Dengan hasil *code* yang kurang akurat, maka *developer* perlu untuk menganalisis kesalahan yang ada di dalam *code* yang dihasilkan, lalu mengimplementasikan perubahan terhadap *code* tersebut.

Upaya untuk mengembangkan penelitian pix2code sudah dilakukan oleh Yanbin Liu, dkk. (2018). Pada penelitian ini dikatakan bahwa penggunaan LSTM untuk melakukan pencarian keterkaitan token DSL dan juga penggabungan vektor kurang efektif karena konten yang berurutan memiliki hirarkinya tertentu. Penelitian Liu dkk. (2018) mengganti metode LSTM dengan *Bidirectional* LSTM (BLSTM) yang ada pada proses pembuatan vektor DSL dan juga pada *decoder*. Penggunaan BLSTM, yang pada dasarnya akan mendapatkan *contextual information* dari kata sebelum dan juga sesudah, dapat meningkatkan hasil/*output* dari proses-proses tersebut. Berdasarkan hasil pengujiannya diperoleh bahwa penggunaan BLSTM menstabilkan proses *training* (ekstrak informasi dari DSL) dengan nilai akurasi *training* set ada pada 92%. Dari segi *output code*, penggunaan BLSTM meningkatkan akurasi menjadi 85% (Liu et al., 2018).

Untuk dapat meningkatkan akurasi, dapat dilakukan dengan memperkaya *dataset*. Namun karena jenis penelitian untuk mengimplementasikan *code* berdasarkan *mockup* merupakan jenis penelitian yang relatif baru, maka *dataset* tersebut sulit didapatkan. Maka dari itu, diperlukan adanya pendekatan lain sehingga akurasi meningkat. Penelitian ini mencoba mengimplementasi *word embedding* pada *language model*, *residual network* pada *vision model*, dan GRU pada *decoder model* untuk meningkatkan nilai akurasi.

II. METODE

Word Embedding

Word embedding atau yang diketahui juga dengan distribusi representasi dari kata, dapat menangkap dan mengukur keterhubungan dari suatu konsep atau kata (Lastra-Díaz et al., 2019). *Word embedding* merepresentasikan kata sebagai vektor padat, yang diturunkan atau didapatkan dengan

menggunakan berbagai macam pelatihan yang terinspirasi dari pemodelan *neural-network language*.

Vektor padat merupakan vektor yang kebanyakan nilainya adalah nilai tak nol, sedangkan kebalikan dari vektor padat adalah vektor renggang (*sparse vector*) yang mana kebanyakan nilainya adalah nilai nol (Levy & Goldberg, 2014).

Penggunaan *one-hot encoding* membuat kata direpresentasikan ke dalam vektor renggang/*sparse vector* sehingga kurang dapat memberikan representasi nilai sebenarnya. Sementara penggunaan *word embedding* dapat menghasilkan vektor yang memiliki fitur yang lebih kompleks dengan menghadirkan data numerik dan nilai kesamaan semantik antar data (Pande Made Risky Cahya Dinatha & Nur Aini Rakhmawati, 2020)

Residual Network

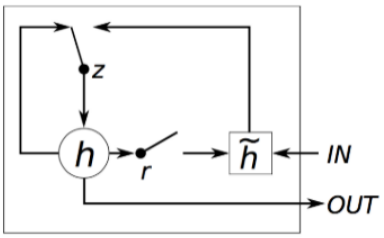
Residual network (ResNet) merupakan arsitektur yang menerapkan fungsi *residual* terhadap *deep learning* yang ada. Arsitektur ResNet akan menggunakan *short-cut* terhadap beberapa *layer* dan tetap menerapkan fungsi *residual*. Sehingga *layer* yang akan dilatih selanjutnya tidak akan kehilangan nilai gradien dari *layer-layer* sebelumnya. Pada salah satu penelitian yang menerapkan ResNet terhadap data uji ImageNet, error yang diberikan hanyalah 3.57% (He et al., 2016). Berdasarkan data-data hasil eksperimen pada penelitian tersebut, telah terbukti penerapan ResNet dapat menurunkan tingkat eror dari sebuah model *deep learning*.

ResNet akan diimplementasikan sebagai arsitektur pada *vision model* karena pada dasarnya arsitektur ResNet dibangun dengan menggunakan CNN. Dalam penelitian ini digunakan arsitektur ResNet50 yang berasal dari *library Keras*.

Gated Recurrent Units (GRU)

GRU serupa dengan LSTM karena keduanya dirancang untuk memecahkan masalah gradien yang hilang pada Recurrent Neural Network (Dey & Salem, 2017). GRU memiliki struktur yang lebih sederhana dibandingkan dengan LSTM. GRU memiliki dua *gate network*, sedangkan LSTM memiliki tiga *gate network*. *Gate network* yang terdapat pada GRU adalah *reset gate* 'r' dan *update gate* 'z'. *Reset gate* memberitahu bagaimana *input* baru akan digabungkan dengan memori sebelumnya. Sedangkan, *update gate* memberi tahu berapa banyak memori sebelumnya yang akan disimpan. Gambar 3 menunjukkan struktur dari GRU. Pada beberapa kasus data *sequence*, GRU terbukti memiliki hasil yang lebih baik dibandingkan dengan LSTM (Hamayel & Owda, 2021; Taneja, 2017).

Oleh karenanya pada penelitian ini GRU akan diterapkan sebagai model *decoder* menggantikan LSTM. Diharapkan dengan mengganti LSTM dengan GRU dapat meningkatkan akurasi dari keseluruhan model.



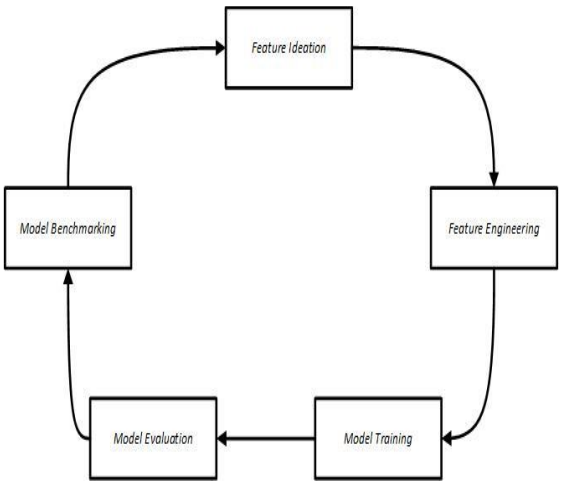
Gambar. 3 Arsitektur GRU (Taneja, 2017)

Dukungan Data

Pada penelitian ini akan digunakan data yang terdapat pada penelitian pix2code Beltramelli (2017). Penelitian ini hanya berfokus pada proses otomasi *code* web saja, maka digunakan 3.500 data berupa gambar *mockup* web dan juga *file* berekstensi '.gui' yang berisikan context *mockup*. Untuk *traning* digunakan 3000 data yang ada, dan untuk *testing* digunakan 500 data. Hal ini mengikuti perbandingan yang dilakukan pada penelitian pix2code (Beltramelli, 2017).

Pengembangan Model

Pengembangan model pix2code dilakukan dengan menggunakan model *agile*, karena pada dasarnya pengembangan model *machine learning* merupakan proses yang *iterative* (Wan et al., 2020). Secara garis besar, proses yang akan dilalui adalah *feature ideation*, *feature engineering*, model *training*, model *evaluation*, dan model *benchmarking*. Gambar 4 menunjukkan alur yang dilalui dalam proses implementasi.



Gambar. 4 Alur Proses Implementasi (Wan et al., 2020)

1) *Feature Ideation:* proses *feature ideation* dilakukan dengan menganalisis *dataset* untuk mendapatkan *feature* yang dibutuhkan untuk kepentingan pengembangan model pix2code.
2) *Feature Engineering: feature engineering* merupakan proses untuk merancang dan membangun *feature* ke dalam *dataset*.
3) Model *Training:* merupakan proses untuk melatih model pix2code sehingga dapat menghasilkan model dengan *weight* yang sudah teroptimasi.
4) Model *Evaluation*: merupakan proses untuk melihat apakah model dapat berjalan dan memenuhi *requirement* atau terdapat *feature* yang dapat di *enhance* guna mengembangkan model lebih lanjut
5) Model *Benchmarking*: merupakan proses untuk membandingkan model dengan suatu nilai tertentu untuk dapat menentukan apakah model bekerja dengan baik atau tidak. Dalam penelitian ini, *model benchmarking* dilakukan dengan melakukan eksperimen terhadap model pix2code yang dikembangkan.

Karena menggunakan model *agile* maka terdapat iterasi dari proses-proses tersebut. Proses pengembangan model pix2code dalam penelitian ini dibagi ke dalam tiga *Milestones*:

1) *Milestones* pertama, mengimplementasikan *word embedding* ke dalam model dan juga perubahan arsitektur model CNN pada *Vision model* menjadi ResNet.
2) *Milestones* kedua, mengimplementasikan GRU kedalam model pix2code
3) *Milestones* ketiga, melakukan perbaikan/*improving* pada model pix2code yang dikembangkan apabila dibutuhkan.

Pengembangan model pix2code dilakukan dengan menggunakan *tools* Google Colab. Karena terdapat batasan waktu dari penggunaan *tools* maka dalam penelitian ini juga diimplementasikan *checkpoint* dalam proses *training* untuk menyimpan data *training* (*loss* & *accuracy*) ke dalam suatu *file* setiap satu *epoch* berakhir. Implementasi *checkpoint* yang dilakukan adalah dengan melakukan pengecekan apabila nilai *loss* yang didapat lebih rendah dibandingkan *epoch* sebelumnya, maka *checkpoint* disimpan, Dengan kata lain data *training* akan disimpan ke dalam suatu *file* untuk kemudian dapat di *load* kembali sehingga proses *training* tidak perlu mengulang dari *epoch* pertama lagi.

Selain mengimplementasikan *checkpoint*, dalam penelitian ini diimplementasikan juga *logging* data *training* yang berfungsi untuk menyimpan data *loss* dan *accuracy* dari setiap *epoch* ke dalam *file* csv. Hal ini dilakukan untuk mempermudah dalam proses analisis model.

Skenario Eksperimen

Eksperimen dilakukan dengan melakukan perubahan terhadap beberapa nilai pada model. Variabel yang diatur dalam proses eksperimen adalah *context length*, *hyperparameter learning rate*, dan *epoch*. Tabel 1 menunjukkan nilai-nilai yang digunakan dalam eksperimen.

Tabel 1. Variabel dalam Eksperimen

Context length	24
	48
	72
Learning rate	0.0001
	0.001
	0.01
Epoch	5
	10

Context length merupakan nilai yang akan mengatur ukuran *context* token yang dipelajari oleh *language model*. *Learning rate* dan *epoch* merupakan variabel yang berupa *hyperparamter* pada model. Pengaturan atau *tuning* terhadap kedua buah variabel ini akan menghasilkan perubahan *output* pada model.

Untuk setiap variabel digunakan nilai optimal pada penelitian pix2code Beltramelli (2017), serta nilai yang lebih kecil dan lebih besar dari nilai optimal tersebut.

Eksperimen akan dilakukan dengan mengombinasikan nilai-nilai tersebut dalam proses *training* model pix2code. Setiap model kemudian akan diuji dengan menghitung nilai akurasi dengan menggunakan tahapan yang didefinisikan dalam proses pengukuran akurasi pada bagian hasil dan pembahasan. Dengan mengkalikan jumlah setiap variabel (3x3x2), didapatkan 18 buah skenario yang dijalankan dalam eksperimen.

III. HASIL DAN PEMBAHASAN

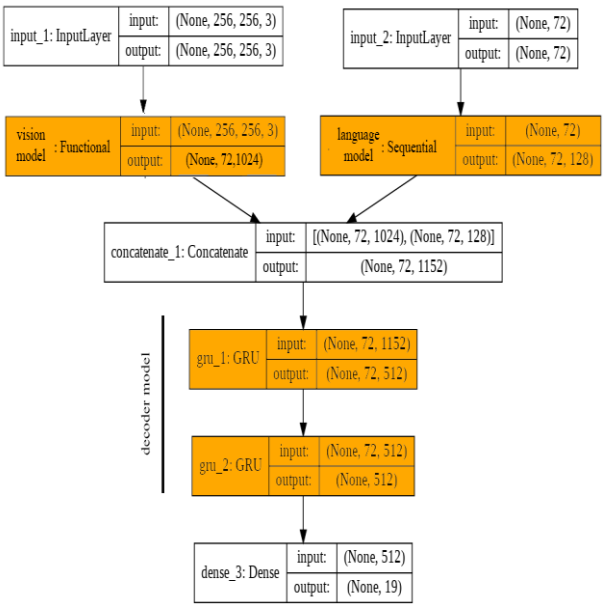
Perubahan Arsitektur Model

Pada dasarnya penggunaan kombinasi *word embedding*, ResNet, dan GRU tidak mengubah arsitektur model pix2code secara signifikan, karena hanya mengubah beberapa *layer* yang digunakan pada model. Di antaranya pada layer dua yang berisi *vision model* dan *language model*, serta layer empat dan lima yang berisi *decoder model*.

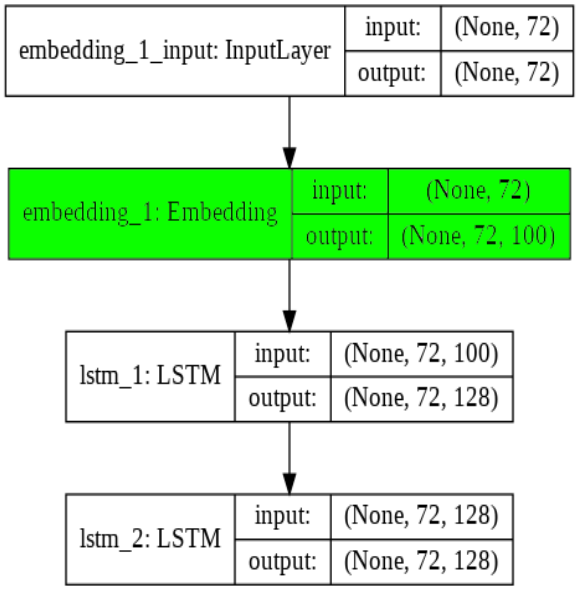
p-ISSN 2477-3506
e-ISSN 2549-1938

Implementasi ResNet50 pada *vision model* dilakukan dengan mengubah implementasi CNN pada model sebelumnya, yaitu dengan melakukan implementasi tiga buah *layer*: ResNet50, GlobalAveragePooling2D, dan Dropout. Implementasi *Word embedding* pada *language model* dilakukan dengan menambahkan embedding *layer* dari *library* Keras ke bagian awal dari *language model*. Implementasi GRU pada *decoder model* dilakukan dengan mengubah implementasi *layer* LSTM pada *decoder model* dengan *layer* GRU.

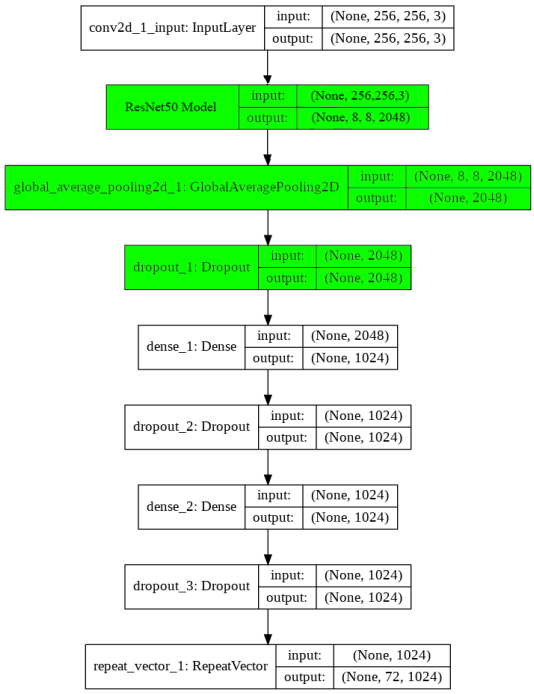
Perubahan pada arsitektur dari pix2code secara keseluruhan ditunjukkan dalam Gambar 5, perubahan pada arsitektur dari *language model* ditunjukkan dalam Gambar 6, dan perubahan pada arsitektur dari *vision model* ditunjukkan dalam Gambar 7. Warna oranye menunjukkan adanya perubahan pada arsitektur sebelumnya, dan warna hijau menunjukkan penambahan *layer* baru pada arsitektur sebelumnya.



Gambar. 5 Perubahan arsitektur model pix2code;



Gbr. 6 Perubahan arsitektur *language model*



Gambar. 7 Perubahan arsitektur *vision model*.

Pengukuran Akurasi

Beltramelli (2017) telah membangun algoritma sederhana untuk mengukur akurasi model pix2code. Untuk memenuhi kebutuhan pengembangan dari model pix2code, maka dilakukan modifikasi atas algoritma tersebut.

Beberapa modifikasi yang dilakukan di antaranya adalah dengan mengubah logika dari penghitungan max_common_length yang menjadi batas untuk *loop*. Selain itu, ditambahkan juga penghitungan min_common_length sehingga dapat diketahui *string* mana dari *groundtruth* dan *generated* yang memiliki nilai minimal dan

maksimal. Berdasarkan nilai tersebut, penghitungan *error* dari perbandingan token dengan menggunakan *loop* dapat dilakukan dengan semestinya. Algoritma (*1*) berikut merupakan algoritma perhitungan akurasi yang telah dimodifikasi.

Algoritma (*1*)

```
Begin

 max_common_length ← max(len(generated_string),
len(ground_truth_string))
 min_common_length ← min(len(generated_string),
len(ground_truth_string))

 error ← error + abs(max_common_length -
min_common_length)
 for i ← 0 to max_common_length do
  if generated_string[i] != ground_truth_string[i] then
   error += 1
  end if
 end for

 error_rate ← error/max_common_length
 accuracy ← 1 – error_rate

 return accuracy
End
```

Algoritma tersebut merupakan algoritma untuk menghitung akurasi dari setiap *file* yang ada dalam *datatest*. Secara matematis formula untuk menghitung akurasi dari setiap *file* ditunjukkan dalam formula (1).

$$Akurasi\ File\ x = 1 - \ Error\ rate\ File\ x \qquad (1)$$

Selanjutnya semua nilai akurasi *file* yang didapatkan akan di rata-rata kan untuk mendapatkan akurasi model secara keseluruhan, formula untuk menghitung akurasi dari setiap model ditunjukkan dalam formula (2).

$$Akurasi\ Model = \frac{\sum_{i=0}^{x} Akurasi\ File\ ke\ i}{Jumlah\ Data\ Test} \qquad (2)$$

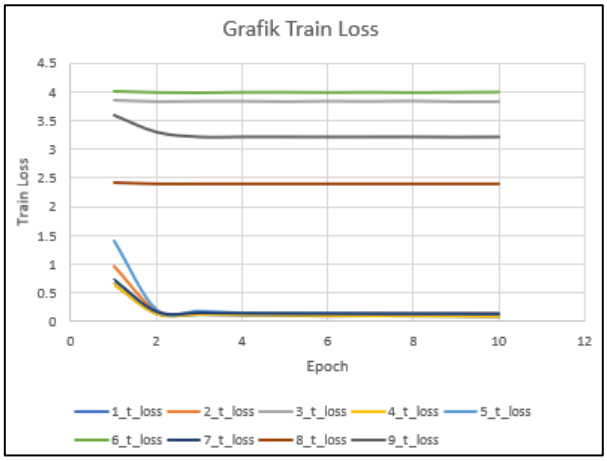
Hasil Eksperimen

Skenario eksperimen dijalankan dengan menggunakan *tools Google Colab*. Proses *training* memakan waktu sekitar satu jam untuk satu *epoch*. Untuk satu buah skenario, keseluruhan proses *training* dan pengukuran akurasi kurang lebih membutuhkan waktu lima jam. Hasil dari eksperimen sesuai skenario di atas dapat dilihat pada Tabel 2.

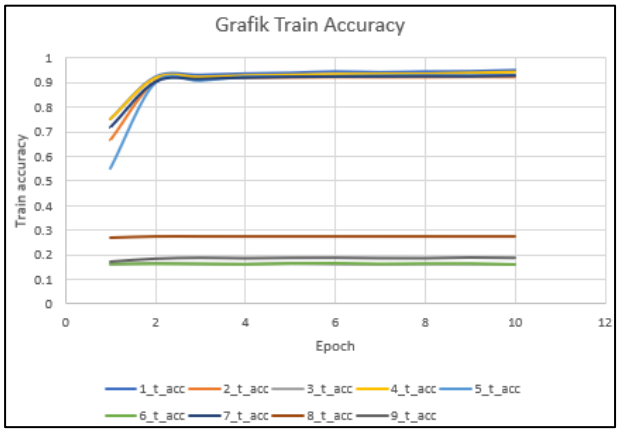
Tabel 2. Hasil Eksperimen Keseluruhan

No. Eksp.	*Context length*	*Learning rate*	*Epoch*	*Mean* Akurasi
1	48	0.001	5	0.0209438088
2	48	0.001	10	0.1076982484
3	48	0.01	5	0
4	48	0.01	10	0.1013066667
5	48	0.0001	5	0.3662867432
6	48	0.0001	10	0.3938267999
7	72	0.001	5	0.1324182408
8	72	0.001	10	0.2625489785
9	72	0.01	5	0
10	72	0.01	10	0
11	72	0.0001	5	0.227368435
12	72	0.0001	10	0.6559309648
13	24	0.001	5	0
14	24	0.001	10	0
15	24	0.01	5	0
16	24	0.01	10	0
17	24	0.0001	5	0.1715702903
18	24	0.0001	10	0.1218502039

Berdasaran eksperimen yang dilakukan, terdapat beberapa kombinasi variabel yang menghasilkan *mean* akurasi 0. Untuk dapat mengetahui penyebabnya, dibangun grafik yang menunjukkan perkembangan nilai akurasi *training* dan *loss training* selama proses *model training* berlangsung. Gambar 8 menunjukkan grafik *loss training* dari keseluruhan eksperimen, sementara Gambar 9 menunjukkan grafik akurasi *training* dari keseluruhan eksperimen.



Gambar. 8 Grafik loss *training* keseluruhan eksperimen



Gambar. 9 Grafik akurasi *training* keseluruhan eksperimen

Berdasarkan Gambar 8 dapat dilihat bahwa terdapat empat buah eksperimen yang memiliki perkembangan yang kurang signifikan selama proses *training* berlangsung. Tiga buah eksperimen yakni 3_t_*loss*, 6_t_*loss*, dan 9_t_*loss* adalah eksperimen yang menggunakan nilai variabel *learning rate* 0.01, pada Tabel 2 eksperimen tersebut memiliki nomor eksperimen 3, 4, 9, 10, 15, dan 16. Satu eksperimen lagi yaitu 8_t_*loss* yang memiliki nilai *loss* yang cukup besar adalah eksperimen dengan nilai *learning rate* 0.001 dan *context length* 24, pada Tabel 2 eksperimen tersebut memiliki nomor eksperimen 13 dan 14. Pada dasarnya, selama proses *training* berlangsung nilai *loss* akan mengalami penurunan mendekati nilai nol. Sementara empat buah eksperimen tersebut tidak menunjukkan penurunan yang signifikan.

Hal yang sama juga terjadi pada grafik akurasi *training* pada Gambar 9, hal ini karena terdapat hubungan antara *loss* dan juga *accuracy training*. Bahwa semakin rendah nilai *loss*-nya maka nilai akurasi *training* juga akan semakin meningkat. Pada umumnya, dalam proses *training*, nilai akurasi pada setiap *epoch*nya akan mengalami peningkatan ke arah nilai 1 (akurasi *training* 100%). Empat buah eksperimen yang menunjukkan performa yang kurang memuaskan pada grafik *loss*, juga menunjukkan hasil yang kurang memuaskan pada grafik akurasi *training*.

Berdasarkan Tabel 2, eksperimen 6 dan 12 menunjukkan dua nilai akurasi terbesar namun masih memiliki nilai yang lebih rendah dibanding akurasi dari model sebelumnya. Sehingga dilakukan eksperimen dalam upaya meningkatkan akurasi dari kedua eksperimen tersebut. Eksperimen lanjutan dilakukan dengan melakukan pertambahan *epoch* dengan jumlah 5 untuk setiap eksperimen yang dilakukan. Hasil eksperimen lanjutan ditunjukkan dalam tabel 3.

Tabel 3. Hasil Eksperimen Tambahan

No. Eksp.	*Context length*	*Learning rate*	*Epoch*	*Mean* Akurasi
6_1	48	0.0001	15	0.5228965194
6_2	48	0.0001	20	0.5361288175
6_3	48	0.0001	25	0.5513567856
12_1	72	0.0001	15	0.2920427856
12_2	72	0.0001	20	0.7209064329
12_3	72	0.0001	25	0.4036642711
12_4	72	0.0001	30	0.5695388373
12_5	72	0.0001	35	0.7644190436
12_6	72	0.0001	40	0.7612856318

Berdasarkan eksperimen tambahan yang dilakukan, eksperimen 12_5 menunjukkan akurasi paling baik dengan nilai 76%. Nilai ini relatif lebih kecil jika dibandingkan dengan nilai akurasi model sebelumnya yaitu 77%. Setelah dilakukan analisis terhadap *code* yang dihasilkan oleh model, ditemukan bahwa dalam beberapa kasus yang memiliki nilai akurasi *file* rendah *code* yang dihasilkan memiliki nilai beda (max-min) yang cukup besar. Hal ini disebabkan karena model menghasilkan token spesial END_TOKEN jauh sebelum *file* tersebut seharusnya berakhir.

Selanjutnya dilakukan modifikasi terhadap implementasi perhitungan akurasi untuk mengabaikan kondisi dari END_TOKEN. Modifikasi terhadap implementasi perhitungan akurasi dengan mengabaikan kondisi END_TOKEN dilakukan dalam upaya untuk menguji performa model dalam memprediksi setiap token. Modifikasi dilakukan dengan tujuan untuk melihat akurasi model tanpa adanya nilai *error* dari selisih panjang *code groundtruth* dan *code* yang dihasilkan oleh model. Modifikasi ini dilakukan murni untuk melakukan eksperimen dengan menguji performa model dalam melakukan generasi setiap token.

Berdasarkan modifikasi yang dilakukan terdapat peningkatan nilai akurasi untuk eksperimen 12_5 dari 76% menjadi 80%. Hal ini menunjukkan bahwa penggunaan kombinasi *word embedding*, ResNet, dan GRU berhasil dalam meningkatkan akurasi dari model sebelumnya.

Selanjutnya, dilakukan juga analisis terhadap hasil distribusi dari akurasi setiap *file* yang dihasilkan. Hal ini dilakukan karena berdasarkan pengamatan terhadap nilai-nilai akurasi yang dihasilkan oleh model pix2code sebelumnya, ditemukan nilai-nilai akurasi *file* yang ekstrim.

Untuk dapat menganalisis distribusi nilai dari akurasi setiap *file*, dihitung nilai standar deviasi dan

variansi dari seluruh nilai akurasi. Hal ini dilakukan untuk dapat melihat distribusi nilai akurasi di sekitar *mean*. Tabel 4 menunjukkan hasil perbandingan yang dilakukan antara model pix2code sebelumnya dengan yang sudah dikembangkan.

Tabel 4. Perbandingan Antar Model

Model Pix2code	*Mean*	Standar Deviasi	Variansi
Pix2code	0.77894	0.280681	0.078782
New Pix2code	0.809403	0.215999	0.046656

Nilai variansi atau standar deviasi dari model pix2code yang dikembangkan memiliki nilai yang lebih kecil jika dibandingkan dengan model pix2code sebelumnya. Hal ini menunjukkan bahwa nilai akurasi yang dihasilkan oleh model pix2code yang dikembangkan memiliki fluktuasi yang lebih rendah dibandingkan model pix2code sebelumnya.

IV. PENUTUP

Kesimpulan

Penelitian ini telah berhasil mengimplementasikan kombinasi dari *word embedding*, ResNet, dan GRU dalam upaya mengembangkan model pix2code. Berdasarkan hasil eksperimen terhadap model baru yang dikembangkan, ditemukan bahwa nilai *learning rate* 0.0001, *context length* 72, dan *epoch* 35 menghasilkan nilai akurasi yang paling besar yakni 76%, namun masih relatif lebih rendah jika dibandingkan akurasi dari model sebelumnya yaitu 77%. Setelah dilakukan analisis lebih lanjut, dilakukan pengembangan metode evaluasi yang berhasil meningkatkan akurasi dari model menjadi 80%.

Selain berhasil meningkatkan akurasi dari model pix2code sebelumnya, penelitian ini juga menemukan bahwa sebaran nilai akurasi masing-masing *file* yang dihasilkan oleh model pix2code sebelumnya memiliki variansi yang lebih besar dibandingkan sebaran nilai akurasi dari model yang dikembangkan di penelitian ini.

Penelitian ini berhasil menurunkan nilai variansi dari 0.078782 menjadi 0.046656. Nilai variansi yang lebih rendah menunjukkan bahwa model pix2code yang dikembangkan dalam penelitian ini memiliki kerapatan nilai akurasi di sekitar *mean* yang lebih baik. Hal ini menunjukkan bahwa nilai akurasi yang dihasilkan oleh model pix2code yang dikembangkan memiliki fluktuasi yang lebih rendah dibandingkan model pix2code sebelumnya.

Saran

Peningkatan akurasi dari penggunaan kombinasi *word embedding*, ResNet, dan GRU masih kurang signifikan, penggunaan kombinasi metode yang digunakan dalam penelitian ini dapat diteliti lebih lanjut lagi dengan melihat bagaimana pengaruh dari masing-masing arsitektur/metode terhadap peningkatan akurasi yang terjadi.

Selain itu, penelitian lanjutan juga dapat dilakukan dengan menerapkan metode ataupun arsitektur lain dalam upaya meningkatkan akurasi dari model. Penerapan *word embedding* pada *language model* dapat dieksplorasi lebih lanjut dengan menggunakan *pre-trained* model *word embedding* seperti word2vec ataupun GloVe. Penerapan arsitektur CNN lain pada *vision model* juga dapat dieksplorasi pengaruhnya, di antaranya seperti arsitektur *inception* atau *exception*.

Selain itu, penelitian lanjutan juga dapat menerapkan elemen-elemen lain yang umumnya ada dalam suatu halaman *website* seperti *input text, radio button*, dll. Pengimplementasian deteksi teks yang ada di dalam *mockup* juga dapat dilakukan untuk dapat mengembangkan model pix2code menjadi lebih *advance* lagi.

V. DAFTAR PUSTAKA

Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., & Tarlow, D. (2017). DeepCoder: Learning to write programs. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*.

Beltramelli, T. (2017). pix2code: Generating Code from a Graphical User Interface Screenshot. *ArXiv*, 1–9.

Dey, R., & Salem, F. M. (2017). Gate-variants of Gated Recurrent Unit (GRU) neural networks. *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 1597–1600. https://doi.org/10.1109/MWSCAS.2017.8053243

Hamayel, M. J., & Owda, A. Y. (2021). A Novel Cryptocurrency Price Prediction Model Using GRU, LSTM and bi-LSTM Machine Learning Algorithms. *AI*, *2*(4), 477–496. https://doi.org/10.3390/ai2040030

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Lastra-Díaz, J. J., Goikoetxea, J., Hadj Taieb, M. A., García-Serrano, A., Ben Aouicha, M., & Agirre, E. (2019). A reproducible survey on word embeddings and ontology-based methods for word similarity: Linear combinations outperform the state of the art. *Engineering Applications of Artificial Intelligence*, *85*, 645–665. https://doi.org/10.1016/j.engappai.2019.07.010

Levy, O., & Goldberg, Y. (2014). Neural word embedding as implicit matrix factorization. *Advances in Neural Information Processing Systems*, *3*(January), 2177–2185.

Liu, Y., Hu, Q., & Shu, K. (2018). Improving pix2code based bi-directional LSTM. *Proceedings of 2018 IEEE International Conference on Automation, Electronics and Electrical Engineering, AUTEEE 2018*, 220–223. https://doi.org/10.1109/AUTEEE.2018.8720784

Pande Made Risky Cahya Dinatha, & Nur Aini Rakhmawati. (2020). Komparasi Term Weighting dan Word Embedding pada Klasifikasi Tweet Pemerintah Daerah. *Jurnal Nasional Teknik Elektro Dan Teknologi Informasi*, *9*(2), 155–161. https://doi.org/10.22146/jnteti.v9i2.90

Patterson, J., & Gibson, A. (2017). Deep Learning A Practioner's Approach. In *O'Reilly Media, Inc.*

Taneja, P. (2017). *Text Generation Using Different Recurrent Neural Networks*. *July*.

Wan, Z., Xia, X., Lo, D., & Murphy, G. C. (2020). How does Machine Learning Change Software Development Practices? *IEEE Transactions on Software Engineering*, 1–14. https://doi.org/10.1109/TSE.2019.2937083