Ekstraksi Partitur Balok Monofonik Untuk Instrumen Flute dengan CRNN dan CRF

Stella Vania[1], Patrick Sutanto[1], Ricky Sutanto[1], dan Joan Santoso[1]
[1]Departemen Informatika, Fakultas Sains dan Teknologi, Institut Sains dan Teknologi Terpadu Surabaya, Surabaya, Indonesia

Corresponding author: Joan Santoso (e-mail: joan@stts.edu).

ABSTRACT Sheet music notation is not easy for beginners to read in the world of music. This is where Optical Music Recognition (OMR) can come into play. OMR is a field of research that investigates how to computationally read musical notation in documents. Music beginners can be helped in reading sheet music with a program that implements OMR and provides output in a format that is easily understood by users. This scientific work is made with a deep learning approach in several architectures. The dataset used is Camera-PrIMuS which consists of an image dataset in a line of sheet music and also the ground-truth per object in the image in question. The architectures used are CRNN, CRNN-CRF, and Attention. Of the three architectures, the best results were obtained for the Attention architecture with a symbol error rate (SER) of around 9%, followed by CRNN with a SER of around 84%, and CRNN-CRF which, based on test results, is not suitable for OMR because of uncoverging loss value. The Attention architecture broadly consists of encoder and decoder blocks. The encoder functions to receive image input and encode the image. The encoding results are then received by the decoder whose role is to perform the decoding and predict the next sequence based on the encoding results from the encoder. In its implementation, the program can receive input in the form of a slightly skewed full sheet sheet image, so the program will also perform skew-correction and crop the image per line so that input from the user can be processed by the model. The output of the model which is still in the form of predictive labels will be processed again to produce numeric notes and MIDI files which are relatively easier for users to understand.

KEYWORDS Artificial Neural Networks, Recurrent Neural Network.

ABSTRAK Notasi partitur balok bukanlah notasi yang mudah dibaca oleh pemula dalam dunia musik. Di sinilah *Optical Music Recognition* (OMR) dapat berperan. OMR merupakan sebuah pembelajaran mengenai komputer yang dapat mengenali objek dalam partitur balok. Dengan adanya program yang menerapkan OMR dan memberikan output dengan format yang mudah dipahami oleh pengguna, maka pemula dalam dunia musik dapat terbantu dalam membaca partitur not balok. Karya ilmiah ini dibuat dengan pendekatan *deep learning* dalam beberapa arsitektur. Dataset yang digunakan adalah Camera-PrIMuS yang terdiri dari dataset gambar sebaris partitur musik dan juga ground-truth per objek pada gambar yang bersangkutan. Arsitektur yang digunakan adalah CRNN, CRNN-CRF, dan *Attention*. Dari ketiga arsitektur tersebut, hasil terbaik diperoleh pada aristektur *Attention* dengan *symbol error rate* (SER) sekitar 9%, diikuti dengan CRNN dengan SER sekitar 84%, dan CRNN-CRF yang berdasarkan hasil uji coba tidaklah cocok untuk OMR dengan nilai loss yang tidak kunjung turun dalam proses training. Arsitektur *Attention* secara garis besar terdiri dari blok *encoder* dan *decoder*. *Encoder* berfungsi untuk menerima input gambar dan melakukan encoding terhadap gambar tersebut. Hasil encoding kemudian diterima oleh decoder yang berperan untuk melakukan *decoding* dan memprediksi *sequence* selanjutnya berdasarkan hasil encoding dari *encoder*. Dalam implementasinya program dapat menerima input berupa gambar selembar partitur penuh yang agak miring, maka program juga akan melakukan *skew-correction* dan pemotongan gambar per baris agar input dari pengguna dapat diproses oleh model. Output dari model yang masih berupa label-label prediksi akan diproses kembali agar menghasilkan not angka dan file MIDI yang relatif lebih mudah untuk dipahami oleh pengguna.

KATA KUNCI Artificial Neural Networks, Recurrent Neural Network

I. PENDAHULUAN

Pemula dalam dunia musik sangat memungkinkan untuk mengalami kesulitan dalam membaca partitur balok. Hal ini wajar mengingat notasi partitur balok tidak begitu mudah untuk dibaca. Dengan demikian, program yang mampu membaca partitur balok dan mengkonversikan hasil pembacaannya dalam bentuk output yang mudah dipahami oleh pengguna dapat membantu para pemula dalam membaca notasi partitur balok.

Optical Music Recognition (OMR) merupakan sebuah bidang penelitian yang bertujuan agar komputer dapat membaca partitur musik [1]. Hasil pembacaan tersebut kemudian akan disimpan ke dalam format yang dapat dibaca mesin. Setelah tersimpan secara digital, informasi musikal tersebut biasanya disimpan dalam format file yang umum sesuai kegunaannya. Sebagai contoh adalah file MIDI jika tujuan dari penerapan OMR itu adalah untuk memainkan kembali musik dari partitur yang bersangkutan.

OMR memiliki kemiripan dengan *Optical Character Recognition* (OCR), keduanya melakukan ekstraksi informasi dari gambar sehingga OMR sempat disebut sebagai music OCR. Walaupun demikian, OMR memiliki tantangan yang berbeda dibanding OCR. Dengan beberapa perbedaan yang ada antara OMR dan OCR, kurang tepat untuk dikatakan bahwa OMR adalah music OCR.

Salah satu perbedaan adalah walaupun notasi partitur balok pada dasarnya tersusun atas elemen-elemen tertentu (kepala not, tangkai not, bendera), posisi elemen tersebut dan kombinasinya terhadap elemen lain dapat mempengaruhi makna semantik dari elemen tersebut. Sebagai contoh, not yang terletak pada garis paling bawah dengan kunci G (nada E4) merepresentasikan nada yang berbeda dengan not pada posisi yang sama namun dengan kunci F (nada G2). Hal seperti demikian tidak ditemui pada permasalahan OCR.

OMR memiliki beberapa pendekatan yang dapat digunakan, salah satunya adalah pendekatan secara *deep learning* yang digunakan pada pembuatan karya ilmiah ini. Karya ilmiah ini dibuat dalam tiga arsitektur, yaitu CRNN, CRNN-CRF, dan *Attention*. Masing-masing arsitektur akan dievaluasi berdasarkan *symbol error rate* yang diperoleh dari perhitungan *Levenshtein distance*. Penggunaan metrik ini terinspirasi dari [2].

II. KAJIAN PUSTAKA

OMR memiliki beberapa pendekatan, baik pendekatan tanpa *machine learning* maupun pendekatan dengan *machine learning*. Secara general, OMR secara garis besar memiliki *pipeline* sebagai berikut, sesuai yang diutarakan [3] dan disempurnakan oleh [4]:

A. PREPROCESSING

Preprocessing merupakan tahapan untuk memanipulasi gambar agar lebih mudah diproses dalam tahapan-tahapan sebelumnya. Yang termasuk dalam tahapan ini antara lain adalah *skew-correction, binarization, noise removal*, dan sebagainya.

B. MUSIC OBJECT DETECTION

Tahapan ini bertujuan untuk menemukan dan mengklasifikasikan objek yang relevan dari gambar.

C. NOTATION ASSEMBLY

Tahapan ini bertujuan untuk memperoleh kembali informasi kontekstual semantik dari objek-objek yang berhasil ditemukan pada tahapan sebelumnya.

D. ENCODING

Tahapan ini bertujuan untuk mengkonversikan informasi kontekstual semantik dari tahapan sebelumnya ke dalam format yang umum sesuai tujuannya. Contohnya adalah format MIDI untuk memainkan musik tersebut, atau MusicXML untuk modifikasi lebih jauh pada program notasi music

Dengan adanya proses pengembangan OMR dengan *machine learning*, beberapa langkah yang biasanya ada dapat dihilangkan atau bahkan menjadi langkah yang lebih besar. Sebagai contoh, tahapan *music object detection* dulunya terdiri dari langkah segmentasi dan langkah klasifikasi. Pada tahapan segmentasi, dilakukan *staff-line removal* karena garis paranada (*staff-line*) dapat mengganggu proses klasifikasi karena garis paranada dapat mempersulit proses pemisahan dua objek yang terpisah dengan menggunakan metode *connected component analysis*. Walaupun demikian, dengan pendekatan *deep learning*, objek tetap dapat terdeteksi tanpa menghilangkan garis paranada, seperti pada [5], [6], [7], [8].

Pada pembuatan karya ilmiah ini, salah satu arsitektur yang dicoba adalah CRNN, seperti yang terdapat pada [2]. Digunakan pula arsitektur CRNN-CRF, CRF pada awalnya diharapkan untuk dapat mempelajari *constraint-constraint* yang diperoleh pada saat training sehingga label yang dipilih bukanlah label dengan probabilitas tertinggi. Dengan adanya CRF, label yang dipilih adalah label dengan probabilitas tertinggi yang memenuhi *constraint* yang dipelajari.

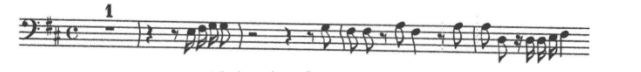
Arsitektur terakhir yang dicoba dalam pembuatan karya ilmiah ini adalah *Attention*. Percobaan arsitektur ini terinspirasi dari cara kerja *neural machine translation* [9][10][13][14]. Uji coba dalam arsitektur ini juga merupakan improvisasi dari arsitektur CRNN. Arsitektur ini dapat memberikan bobot lebih (*attention*) pada bagian dari partitur yang berkaitan dengan suatu label tertentu. Secara garis besar, arsitektur ini terdiri dari *encoder* dan *decoder*. Layer *encoder* terdiri dari layer CNN[11] dan Bi-LSTM. Layer *decoder* terdiri dari layer LSTM[15]. Jika diperhatikan, layer-layer yang digunakan pada arsitektur *Attention* memiliki kemiripan dengan layer CRNN yang terdiri dari layer CNN dan Bi-LSTM. Walaupun demikian,

perbedaan proses yang terjadi di antara layer-layer tersebut memberikan hasil akhir yang berbeda. Pada arsitektur *Attention*, output LSTM pada *decoder* tidak langsung digunakan untuk memprediksi, melainkan diproses dengan output Bi-LSTM dari *encoder* untuk memperoleh bobot *attention*. Dengan adanya mekanisme seperti ini, arsitektur ini dapat melakukan *aligning* antara gambar dengan label secara otomatis.

III. *DATASET*

Dataset yang digunakan dalam pengerjaan penelitian ini berjudul "*Camera Printed Images of Music Staves*" (Camera-PrIMuS). Dataset ini digunakan dan diterbitkan bersama dengan paper *Camera-PrIMuS: Neural End-To-End Optical Music Recognition on Realistic Monophonic Scores* [2]. Dataset ini diterbitkan dengan tujuan untuk memenuhi kebutuhan dataset dalam penelitian *Optical Music Recognition* (OMR) dengan pendekatan *machine learning*, dikarenakan seluruh penelitian berbasis *machine learning* membutuhkan dataset dalam jumlah besar dengan kualitas baik.

Dataset Camera-PrIMuS terdiri dari 87.678 awalan partitur not balok real yang direpresentasikan dalam enam jenis format file. Pada pembuatan karya ilmiah ini, hanya akan digunakan dua jenis format file dari keenam format yang ada, yaitu gambar partitur terdistorsi dengan format jpg dan representasi simbol musik dalam *semantic encoding* sebagai *ground-truth*. Contoh dari kedua format data yang digunakan dapat dilihat pada gambar 1.



(a) Gambar partitur musik

clef-F4 keySignature-DM timeSignature-C multirest-1 barline rest-quarter rest-eighth note-E3_sixteenth
note-F#3_sixteenth note-G3_sixteenth note-G3_eighth barline rest-half rest-quarter rest-eighth note-G3_eighth
barline note-F#3_eighth note-F#3_eighth rest-eighth note-A3_eighth note-F#3_quarter rest-eighth note-A3_eighth
barline note-A3_eighth note-D3_eighth rest-sixteenth note-D3_sixteenth note-D3_sixteenth note-E3_sixteenth
note-F#3_quarter

(b) Semantic encoding

GAMBAR 1. Contoh sampel dataset

Pada *semantic encoding* dataset Camera-PrIMuS, terdapat 1.781 label. Secara garis besar, 1.781 label tersebut terdiri dari 10 label *clef*, 14 label *key signature*, 39 label *time signature*, 1384 label note dengan *range* nada antara A2 hingga G5, 210 label *grace* note dengan *range* nada antara A2 hingga G5, 24 label *rest*, 98 label *multirest*, 2 label lainnya, yaitu *tie* dan *barline*.

Pada notasi partitur not balok real, seluruh objek-objek musikal tidak memiliki jumlah kemunculan yang sama. Karena dataset Camera-PrIMuS dibuat berdasarkan notasi musik *real*, persebaran kemunculan tiap label pun juga tidak merata. Sebagai contoh, label berjenis *note* jauh lebih sering muncul dibanding label jenis lainnya. Hal ini sangatlah wajar karena mayoritas bagian dari notasi musik tersusun atas note. Label *barline* menduduki urutan kedua, karena setiap data pasti memiliki beberapa barline.

A. PREPROCESSING GAMBAR

Walaupun gambar partitur pada dataset yang berformat jpeg memiliki tiga *color channel*, gambar akan diload secara grayscale. Proses *load* gambar secara grayscale ini dilakukan dengan pertimbangan bahwa partitur pada umumnya memiliki warna yang bersifat monokrom. Pixel-pixel gambar direpresentasikan dalam bentuk matrix dua dimensi. Setiap pixel memiliki range nilai antara 0 hingga 255, semakin terang warna yang direpresentasikan pixel tersebut, semakin besar pula nilainya. Nilai-nilai tersebut akan dibalik dengan mengurangkannya dengan 255. Nilai-nilai tersebut akan dinormalisasi sehingga menjadi berkisar antara 0 hingga 1.

Model membutuhkan input gambar dengan dimensi yang sama untuk setiap *batch*-nya, sementara gambar partitur pada dataset memiliki dimensi lebar dan tinggi yang bervariasi. Karena itulah proses *resize* perlu dilakukan. Walaupun demikian, proses *resize* tidak dapat dilakukan dengan menentukan ukuran yang pasti untuk setiap gambar, karena setiap gambar memiliki proporsi panjang dan tinggi yang berbeda. Jika dipatok ukuran yang sama untuk setiap gambar, maka akan terdapat gambar yang *stretch*, gambar seperti demikian dapat mengganggu proses training.

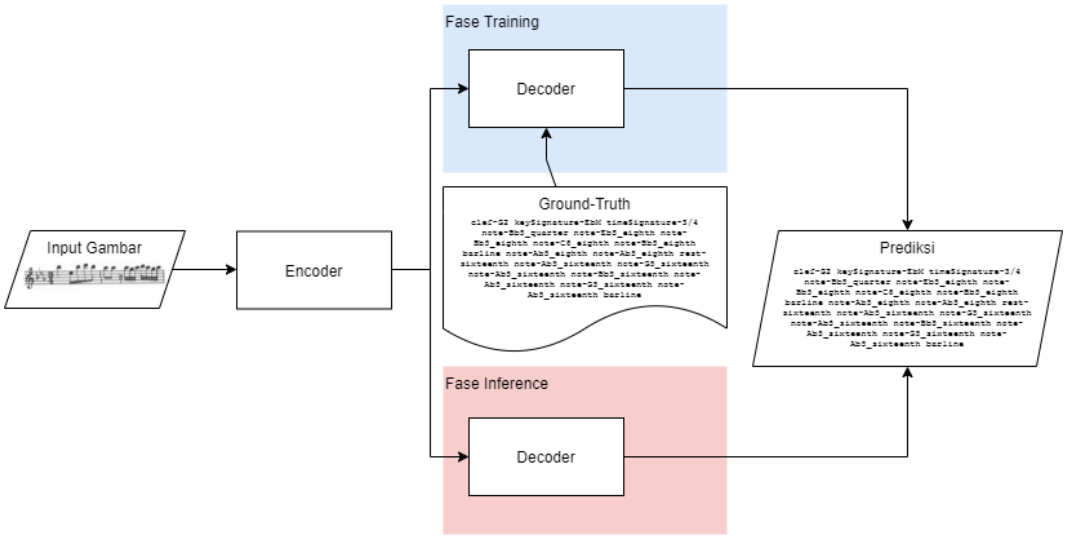
Agar gambar tidak stretch, proses resize yang dilakukan adalah sebagai berikut. Pertama, akan dicari terlebih dahulu dimensi terpanjang dan tertinggi pada batch yang diproses. Memperbesar gambar dari ukuran aslinya dapat memperburuk kualitas gambar. Gambar berukuran besar juga dapat memperlambat proses training karena jumlah parameter yang akan di-training juga semakin banyak. Dengan alasan ini, kedua nilai tersebut masing-masing diperkecil menjadi 60 persennya saja.

Dalam proses *resize*, tinggi gambar akan menjadi sesuai dengan tinggi maksimal gambar pada batch tersebut. Untuk menjaga proporsi gambar, panjang gambar setelah proses *resize* diperoleh dengan membagi panjang dengan tinggi gambar asli dan mengalikannya dengan dimensi tinggi maksimal. Agar dimensi panjang seluruh gambar pada *batch* juga seragam, maka gambar yang tidak mencapai dimensi panjang maksimal akan diberi padding sehingga panjangnya menjadi seragam, yaitu 60% dari panjang maksimal gambar pada batch yang diproses

B. PREPROCESSING GROUND-TRUTH

Pada dataset Camera-PrIMuS, *ground-truth* untuk setiap data gambar adalah berupa rangkaian label untuk setiap objek yang terdapat pada partitur tersebut. Setiap label pada suatu file *ground-truth* untuk masing-masing gambar dipisahkan dengan tab.

Preprocessing ground-truth diawali dengan penambahan token "<START>" pada awal setiap rangkaian ground-truth



GAMBAR 2. Arsitektur Model Secara Garis Besar

dan juga token "<END>" pada akhir setiap rangkaian ground-truth. Penambahan kedua token tersebut berfungsi sebagai penanda awal dan akhir sequence untuk model yang menerapkan konsep *attention*, hal ini dilakukan karena *decoder* dalam fase *training* pada *attention* membutuhkan kedua penanda tersebut

Sama seperti pada gambar, model juga membutuhkan *ground-truth* dengan panjang yang sama per *batch*-nya. Maka dari itu, akan dihitung rangkaian *ground-truth* terpanjang pada *batch* yang diproses. Setiap rangkaian *ground-truth* yang lebih pendek akan diberi token "<PAD>" seperlunya setelah token "<END>" hingga memiliki panjang yang sama dengan rangkaian *ground-truth* terpanjang

Pada akhir *preprocessing ground-truth*, seluruh *ground-truth* beserta token-token "<START>", "<END>", dan "<PAD>" akan diubah ke dalam bentuk *one-hot encoding*. Proses perubahan ke bentuk ini didasari dengan pemahaman bahwa model tidak dapat secara langsung menangani label-label tersebut sehingga label-label tersebut perlu diubah ke dalam bentuk representasi angka. Representasi angka dalam bentuk *one-hot encoding* dipilih karena dalam dataset Camera-PrIMuS, label-label yang ada tidak bersifat *ordinal* (tidak memiliki urutan khusus). Berbeda kasusnya jika label-label ground-truth memiliki urutan berdasarkan label-label itu sendiri. Contoh untuk kasus ini adalah label dingin, normal, dan panas.

Dalam kasus dengan label ini memiliki urutan tersendiri berdasarkan suhunya, representasi angka untuk label dapat digantikan dengan representasi angka biasa secara berurutan.

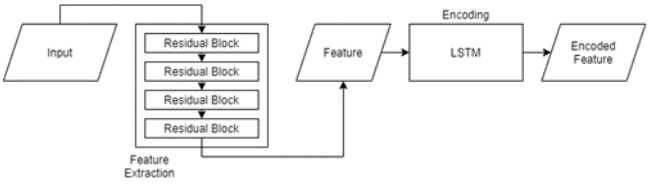
IV. METODE YANG DIGUNAKAN

Arsitektur yang yang dibuat pada karya ilmiah ini berbasiskan konsep *attention* yang secara garis besar terdiri atas *encoder* dan *decoder*. Gambaran mengenai arsitektur yang diajukan dapat dilihat pada **Error! Reference source not found.**. Tujuan dari arsitektur ini adalah untuk melakukan prediksi label suatu objek tertentu dari suatu bagian gambar berdasarkan probabilitas tertingginya.

$$\hat{y} = \arg\max P(y|x) \quad (1)$$

A. ENCODER

Blok *encoder* berperan dalam melakukan *embedding* terhadap gambar. *Encoder* yang dibuat pada penelitian ini secara garis besar terdiri dari empat *residual block*[12] dan sebuah layer LSTM. *Residual block* yang pertama menerima input berupa gambar yang telah melalui proses *preprocessing*. Output dari setiap *residual block* akan diteruskan ke *residual block* selanjutnya. Kegunaan utama dari *residual block* adalah untuk melakukan *feature extraction*. *Feature* yang diperoleh akan melalui proses *encoding* pada layer LSTM menjadi *encoded feature*. Untuk lebih jelasnya, dapat dilihat pada gambar 3.



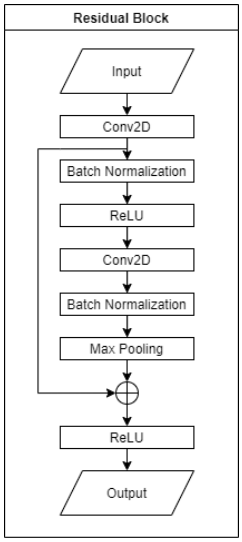
GAMBAR 3. Arsitektur Encoder

Encoder menerima input gambar per *batch* yang telah melalui *preprocessing*. Pada pengerjaan penelitian ini, jumlah data per batch adalah 16. Masing-masing gambar pada batch terdiri dari satu *channel* warna. Setiap gambar memiliki 2 dimensi, yaitu dimensi panjang dan tinggi

Walaupun ukuran panjang dan tinggi gambar pada suatu *batch* harus sama, masing-masing *batch* dapat memiliki dimensi panjang dan tinggi yang berbeda-beda. Penentuan

dimensi panjang dan tinggi gambar pada suatu batch dilakukan berdasarkan dimensi gambar terpanjang dan dimensi gambar tertinggi dalam batch yang bersangkutan.

Untuk memperoleh feature yang akan di-train pada proses *training*, perlu dilakukan *feature extraction* untuk mengambil *feature* dari gambar input. Proses *feature extraction* ini dilakukan oleh *residual block*. Pada penelitian ini, digunakan empat *residual block*, output suatu *residual block* akan menjadi input untuk *residual block* selanjutnya. Output dari *residual block* terakhir adalah berupa *feature* yang akan diumpankan ke layer selanjutnya pada *encoder*, yaitu layer LSTM. Arsitektur masing-masing *residual block* pada penelitian ini dapat dilihat pada.gambar 4



GAMBAR 4. Arsitektur Residual Block

Pada masing-masing residual block, input pada awalnya akan melalui *convolutional* layer 2 dimensi. Karena adanya distribusi data yang berbeda pada setiap *batch*, maka perlu dilakukan normalisasi. Tanpa adanya normalisasi, proses training akan menjadi kurang optimal karena karena model menjadi kurang stabil. Untuk itulah output *convolutional* 2 dimensi pertama melewati layer *batch normalization*[16].

Setelah melalui layer *batch normalization*, terdapat activation function ReLU[19][20] yang membiarkan nilai positif dan mengubah seluruh nilai negatif menjadi 0. Dengan adanya ReLU, tidak terdapat lagi nilai negatif pada output.

Nilai yang telah melalui activation function ReLU tersebut akan melalui *convolutional* layer 2 dimensi dan *batch normalization* layer lagi. Untuk mengurangi *feature* yang dihasilkan, digunakanlah layer *max pooling*[18].

Untuk menghindari permasalahan *vanishing gradient* ataupun *exploding gradient*[17] yang umum terjadi dengan banyaknya layer yang digunakan, output dari seluruh rangkaian sebelumnya ditambahkan dengan output dari *convolutional* layer 2 dimensi pertama. Dengan demikian, *feature-feature* yang mungkin mulai menghilang ataupun terlalu menonjol dapat distabilkan kembali. *Feature* tersebut akan diteruskan ke *residual block* selanjutnya untuk diproses. Pada *residual block* terakhir, *feature* akan diteruskan menuju layer LSTM.

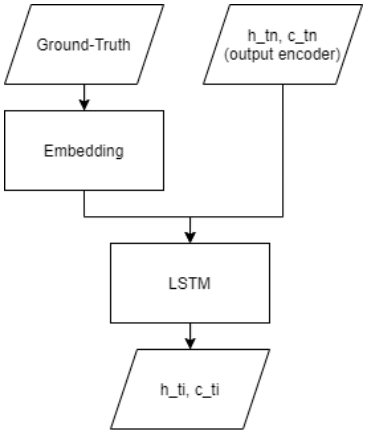
Layer LSTM pada *encoder* berperan dalam melakukan *encoding* terhadap *feature* yang dihasilkan *residual block* sehingga *encoded feature* tersebut dapat diproses lebih lanjut oleh *decoder*. Lebih spesifiknya, layer LSTM yang digunakan pada pengerjaan penelitian ini adalah Bi-LSTM.

Pada pengerjaan penelitian ini, digunakan hanya satu layer LSTM saja. Layer LSTM pada awalnya akan menerima input berupa *feature* dari *residual block*. *Hidden state* dan *cell state* pertama diinisialisasi dengan 0, nilai-nilai tersebut akan berubah seiring dengan proses yang berjalan. Output dari LSTM adalah *hidden state* dari masing-masing *timestep*, *hidden state* terakhir, dan *cell state* terakhir

B. DECODER

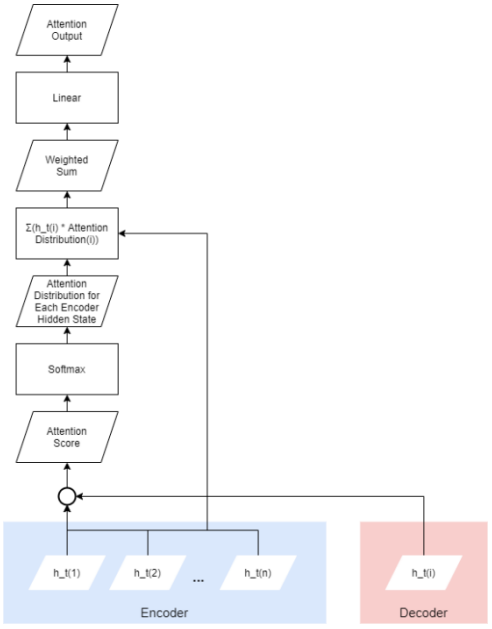
Decoder memiliki cara kerja yang berbeda pada fase *training* dan *inference*. Walaupun demikian, pada fase *training* maupun fase *inference*, *decoder* selalu menerima input yang merupakan output dari layer LSTM pada blok *encoder*. *Decoder* akan menerima *hidden state* dari setiap *timestep* pada LSTM *encoder, hidden state* terakhir, dan juga *cell state* terakhir. Sama seperti pada blok *encoder*, blok *decoder* juga melakukan prosesnya per *batch*.

Pada fase *training, decoder* tidak hanya menerima input berupa *hidden state* dan *cell state* terakhir dari blok encoder, *decoder* juga menerima rangkaian *ground-truth* yang akan diumpankan secara berurutan pada setiap *timestep*. *Ground-truth* yang diumpankan akan melalui proses *embedding* terlebih dahulu agar *ground-truth* dengan relasi yang erat memiliki representasi yang mirip. *Ground-truth* yang telah melalui proses embedding akan diumpankan ke layer LSTM bersama dengan *hidden state* dan *cell state* terakhir dari blok encoder. Output dari LSTM adalah *hidden state* dan *cell state* untuk *timestep* tersebut.



GAMBAR 5. Diagram Alur Proses Seputar LSTM pada Decoder Fase Training Dalam Satu Timestep

Hidden state yang dihasilkan pada suatu *timestep* pada LSTM *decoder* akan "dihubungkan" dengan masing-masing *hidden state* dari LSTM *encoder*. Proses penghubungan inilah yang menjadi inti dari *attention*. Proses penghubungan ini dilakukan dengan perkalian matrix. Hasil dari perkalian matrix ini adalah *attention score*, yaitu bobot yang menentukan seberapa besar keterkaitan antara *ground-truth* dengan *feature* yang terdapat pada *encoder*. *Attention score* ini akan dilewatkan ke *activation function softmax* sehingga diperoleh distribusi *attention* untuk masing-masing *hidden state encoder* dengan jumlah seluruh nilai distribusi tersebut adalah 1. *Hidden state* dari *encoder* dengan keterkaitan yang tinggi dengan *hidden state decoder* pada *timestep* tersebut akan memperoleh distribusi *attention* yang lebih banyak dibanding dengan *hidden state encoder* lainnya.



GAMBAR 6. Arsitektur Attention pada Decoder Fase Training Dalam Satu Semester

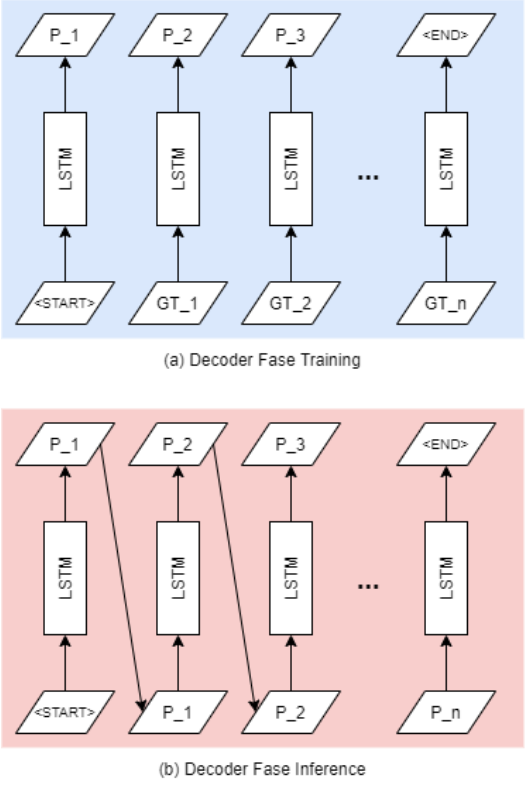
Attention akan diaplikasikan dengan mengalikan distribusi *attention* tersebut dengan masing-masing nilai *hidden state* yang bersangkutan. Dari sinilah akan diperoleh nilai *hidden state* dari setiap *timestep* pada fase *encoder* yang telah memiliki bobot (*weighted sum*). Nilai ini akan dilewatkan pada layer *linear* untuk melakukan *mapping* untuk menyesuaikan ukurannya tanpa menghilangkan informasi kontekstual yang dikandungnya. Hasil dari *linear* layer ini adalah *attention* output, yaitu prediksi untuk *timestep* yang bersangkutan. Prediksi dari masing-masing *timestep* akan dibandingkan dengan *ground-truth* pada perhitungan *loss*.

Pada pengerjaan penelitian ini, terdapat banyak label yang memungkinkan untuk masing-masing objek dengan masing-masing objek hanya dapat memiliki satu class saja. Dengan kasus seperti ini, *loss function* yang digunakan adalah *categorical cross-entropy*. Rumus untuk *categorical cross-entropy* adalah sebagai berikut:

$$Loss = -\sum_{i=1}^{n} y_i \cdot \log \hat{y}_i \qquad (2)$$

Dengan y_i adalah ground-truth dan $\hat{y}_i$ adalah output attention (hasil prediksi) pada *sequence* ke-i. Karena akan terdapat sequence sepanjang n, maka akan terbentuk vektor sepanjang n dengan masing-masing nilainya merupakan nilai *loss* per object. Untuk memperoleh nilai *loss* final, diambil nilai rata-rata dari seluruh loss pada vektor tersebut. *Decoder* pada fase inference masih menggunakan layer LSTM yang sama seperti LSTM pada *decoder* fase *inference*. Kegunaan dari layer LSTM pada decoder fase *inference* juga masih sama seperti pada fase *training*, yaitu untuk melakukan proses *decoding* dari *feature* yang diumpankan.

Pada fase *inference*, proses yang dilakukan tidak jauh berbeda. Hal yang membedakan adalah LSTM pada fase *inference* dari fase training menerima input yang berbeda. Layer LSTM pada *decoder* fase *inference* menerima input berupa prediksi dari *timestep* sebelumnya, *hidden state* terakhir dari *encoder*, dan *cell state* terakhir dari *encoder*. Prediksi dari *timestep* sebelumnya akan melalui proses *embedding* terlebih dahulu sebelum diproses lebih lanjut. *Embedding* prediksi *timestep* sebelumnya akan diumpankan ke layer LSTM bersama dengan *hidden state* dan *cell state* terakhir dari blok encoder. Output dari LSTM adalah *hidden state* dan *cell state* untuk *timestep* tersebut.



GAMBAR 7. Ilustrasi Perbandingan Alur Informasi pada Decoder Fase Training dengan Fase Inference

V. UJI COBA

A. EVALUATION METRICS

Untuk menilai apakah model yang dibuat sudah baik atau belum, perlu dilakukan pengukuran seberapa berbeda sequence hasil prediksi model dengan ground-truth-nya. Evaluation metrics yang cocok untuk permasalahan ini adalah *Levenshtein distance, Levenshtein distance* merupakan metrik untuk mengukur perbedaan antara dua sequence. Nilai *Levenshtein distance* adalah jumlah pengubahan (penyisipan, penghapusan, dan penggantian) minimal untuk mengubah suatu *sequence* agar menjadi serupa dengan sequence satunya. *Levenshtein distance* dirumuskan sebagai berikut:

$$lev_{(a,b)} = \begin{cases} max(i,j), & jika\ min(i,j) = 0 \\ min \begin{cases} lev_{(a,b)}(i-1,j)+1 \\ lev_{(a,b)}(i,j-1)+1 \\ lev_{(a,b)}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} \end{cases} \quad (3)$$

Dengan:

- a dan b merupakan masing-masing *sequence*
- i dan j merupakan posisi index pada masing-masing *sequence*

Nilai *Levenshtein distance* masih perlu diproses kembali untuk mengekspresikan seberapa baiknya performa model dalam bentuk persentase. Untuk itu, nilai Levenshtein *distance* akan dinormalisasi dengan panjangnya dan dirata-rata per batch untuk memperoleh *symbol error rate*

B. HASIL UJI COBA

Model dengan arsitektur *attention* pada penelitian ini di-*train* per *mini-batch*. Training dilakukan hingga 16.000 epoch, dalam setiap epoch, satu *mini-batch* berisi 16 data diumpankan. Hasil dari *training* model dapat dilihat pada tabel I

TABEL I
HASIL UJI COBA ARSITEKTUR ATTENTION PER 2.000 EPOCH

Model	Epoch	Loss	Symbol Error Rate (%)		
			Train	Test	Val
Attention-A	2.000	0.87	56.67%	57.11%	56.63%
Attention-B	4.000	0.46	26.29%	27.38%	26.79%
Attention-C	6.000	0.27	20.13%	21.59%	21.10%
Attention-D	8.000	0.28	15.50%	16.95%	16.79%
Attention-E	10.000	0.20	13.08%	14.72%	14.51%
Attention-F	12.000	0.13	11.58%	13.43%	13.18%
Attention-G	14.000	0.11	8.30%	10.14%	9.86%
Attention-H	16.000	0.07	7.94%	9.68%	9.57%

Perbedaan tiap arsitektur pada table 1 terletak pada jumlah training(epoch).Berdasarkan tabel di atas, dapat dilihat bahwa model Attention-A memiliki nilai loss yang besar, yaitu 0.87. Ketika model tersebut dievaluasi dengan digunakan untuk memprediksi data pada dataset, diperoleh hasil *symbol error rate* pada data di dataset training dengan nilai 56.67%. *Symbol error rate* yang diperoleh model Attention-A ketika memprediksi data pada *test set* sedikit lebih tinggi, yaitu 57.11%. Hal yang menarik ditemukan ketika model tersebut digunakan untuk memprediksi data pada *validation set*, karena *symbol error rate* yang diperoleh justru sedikit lebih rendah dibanding dengan *symbol error rate* pada data training.

Model Attention-B yang di-training dengan 4.000 epoch menghasilkan nilai *loss* yang jauh lebih rendah dibanding dengan model Attention-A. Loss yang dihasilkan mengecil hingga kurang lebih setengah dari *loss* pada model sebelumnya. Rata-rata *symbol error rate* pada model Attention-B pun juga mengecil hingga sekitar setengah dari Attention-A. Pada Attention-B, tetap terlihat bahwa hasil evaluasi pada test set memiliki *symbol error rate* tertinggi. Walaupun demikian, hasil evaluasi pada *train set* dalam Attention-B memiliki *error rate* yang lebih kecil dibanding pada *validation set*.

Model Attention-C di-training dengan 6.000 epoch dan menghasilkan nilai loss yang secara signifikan lebih rendah dibanding dengan Attention-A dan Attention-B. Ketika dievaluasi, *symbol error rate* juga ikut mengecil pada ketiga bagian dari dataset yang dicoba. Hasil evaluasi dengan train set pada Attention-C menghasilkan *symbol error rate* sekitar 1% lebih rendah dibanding dengan set lainnya.

Model Attention-D di-training dengan 8.000 epoch. Walaupun nilai *loss* yang dihasilkan sedikit lebih tinggi dibanding dengan Attention-C, *symbol error rate* yang muncul dalam tahap evaluasi tetap mengecil hingga sekitar 5% dari Attention-C. Tetap terlihat bahwa hasil evaluasi pada train set memiliki hasil terbaik dengan nilai *symbol error rate* terkecil dan hasil evaluasi pada *test set* memperoleh hasil terburuk.

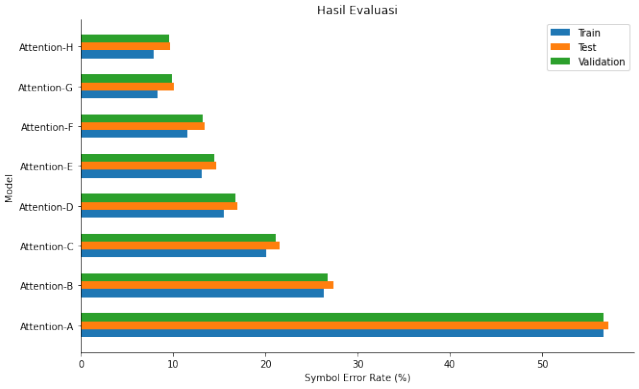
Pada model Attention-E yang di-training dengan 10.000 epoch, nilai loss terlihat mengecil. Hasil evaluasi pada train, test, dan validation set juga turut mengecil walaupun penurunan *symbol error rate* sudah tidak sesignifikan sebelumnya. Hasil evaluasi pada *train set* tetap mengungguli set lainnya, perbedaan tersebut menjadi semakin besar, yaitu sekitar 1,5% dibanding dengan *test set*.

Model Attention-F yang di-training dengan 12.000 epoch menghasilkan nilai *loss* yang semakin mengecil. Walaupun demikian, penurunan *symbol error rate* pada tahap evaluasi tidak begitu besar. Ketiga set rata-rata menghasilkan *symbol error rate* sekitar 12%. *Test set* tetap menghasilkan *symbol error rate* tertinggi, diikuti dengan *validation set* dan *train set.*

Model Attention-G menghasilkan *loss* yang agak mengecil dibanding dengan model sebelumnya setelah di-train dengan 14.000 epoch. Pada model Attention-G, hasil evaluasi yang diperoleh terlihat membaik dibanding model-model sebelumnya dengan rata-rata sekitar 9%.

Model Attention-H melalui proses training dengan 16.000 epoch. Karena *loss* yang dihasilkan dianggap sudah cukup kecil, yakni kurang dari 0,1, dan penurunan *loss* sudah tidak begitu signifikan yang berarti *loss* sudah cukup konvergen,

maka model inilah yang digunakan. Ketika dilakukan tahapan evaluasi, terlihat pula bahwa *symbol error rate* pada masing-masing set juga menurun dibanding dengan hasil evaluasi untuk model Attention G. Perbedaan *symbol error rate* antara *train set* dan *test set* terlihat semakin membesar pula, yakni hampir mencapai 2%



GAMBAR 8. Grafik Visualisasi Hasil Evaluasi untuk Masing-Masing Model Attention

Dari seluruh uraian di atas, dapat ditarik kesimpulan sementara bahwa performa model meningkat pada jumlah epoch yang lebih banyak. Di antara model Attention-A yang memiliki 2.000 epoch dan Attention-B yang memiliki 4.000 epoch, terdapat penurunan *symbol error rate* yang signifikan. Penurunan terus terjadi di antara model-model selanjutnya hingga Attention-H. Di antara model Attention-G dan Attention-H, masih terlihat adanya penurunan yang sangat kecil sehingga disimpulkan bahwa model sudah cukup konvergen. Hasil evaluasi pada train set hampir selalu memberikan *symbol error rate* terendah, hal ini wajar mengingat model di-*train* dengan menggunakan *train set*

VI. KESIMPULAN

Arsitektur CRNN menghasilkan *symbol error rate* yang sangat tinggi, yaitu 84,8%. Walaupun demikian, nilai *loss* pada awal training hingga akhir proses training mengalami penurunan yang signifikan.

Nilai loss pada arsitektur CRNN-CRF dalam proses training sangat besar dan tidak mengalami penurunan setelah di-*train* dalam 4.000 epoch hingga *training* akhirnya dihentikan. Dari sini dapat disimpulkan bahwa arsitektur ini tidak cocok dalam menangani permasalahan ini.

Arsitektur *Attention* menghasilkan nilai loss terus yang menurun hingga batas tertentu pada proses training. Ketika dievaluasi, *symbol error rate* yang dihasilkan berkisar antara 10%. Dengan demikian dapat disimpulkan bahwa arsitektur *Attention* merupakan arsitektur yang paling cocok untuk menangani permasalahan ini dibanding dengan kedua arsitektur lainnya.

Skew-correction yang diterapkan untuk menangani input pengguna sebelum diproses oleh model dapat berjalan dengan baik jika warna kertas kontras dengan warna latar belakangnya. *Skew-correction* juga berjalan optimal apabila tidak terdapat motif pada latar belakang gambar yang dapat mengganggu proses deteksi kertas yang terdapat dalam *skew-correction*.

Pemotongan gambar partitur per baris dilakukan dengan melakukan dilasi berulang kali terhadap hasil *edge-detection* pada gambar. Proses ini tidak dapat berjalan optimal terdapat bagian dari partitur yang blur karena dapat mempengaruhi proses *edge-detection*. Selain itu, *noise* yang terdapat pada gambar juga dapat memperburuk performa dari pemotongan ini karena dapat ikut terdeteksi sebagai baris partitur setelah hasil *edge-detection* didilasi.

File MIDI yang dihasilkan oleh program dapat memainkan nada-nada dengan ketukan yang benar sesuai hasil prediksi. Hasil yang baik ini dapat diperoleh dengan mudah mengingat format label yang seragam sehingga mempermudah proses parsing dan konversi ke file MIDI.

Seperti halnya pada file MIDI, not angka yang dihasilkan juga sesuai dengan hasil prediksi model, output model setelah melalui *postprocessing* adalah label-label dengan format yang seragam sehingga mempermudah proses *parsing* dan konversi ke not angka

COPYRIGHT

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

DAFTAR PUSTAKA

[1] Jorge Calvo-Zaragoza, Jan Hajič Jr., and Alexander Pacha. 2020. Understanding Optical Music Recognition. ACM Comput. Surv. 53, 4, Article 77 (September 2020)

[2] Jorge Calvo-Zaragoza, & David Rizo. (2018). Camera-PrIMuS: Neural End-to-End Optical Music Recognition on Realistic Monophonic Scores. Proceedings of the 19th International Society for Music Information Retrieval Conference, 248–255

[3] David Bainbridge and Tim Bell. The challenge of optical music recognition. Computers and the Humanities, 35(2):95–121, 2001.

[4] Ana Rebelo, Ichiro Fujinaga, Filipe Paszkiewicz, Andre R.S. Marcal, Carlos Guedes, and Jamie dos Santos Cardoso. Optical music recognition: state-of-the-art and open issues. International Journal of Multimedia Information Retrieval, 1(3):173–190, 2012.

[5] J. Calvo-Zaragoza and D. Rizo. End-to-End Neural Optical Music Recognition of Monophonic Scores. Applied Sciences, 8(4):606–629, 2018.

[6] J. Hajic Jr. and P. Pecina. Detecting Noteheads ˇ in Handwritten Scores with ConvNets and Bounding Box Regression. Computing Research Repository, abs/1708.01806, 2017

[7] A. Pacha, K.-Y. Choi, B. Couasnon, Y. Ricquebourg, ¨ R. Zanibbi, and H. Eidenberger. Handwritten music object detection: Open issues and baseline results. In 13th IAPR Workshop on Document Analysis Systems, pages 163–168, 2018

[8] E. van der Wel and K. Ullrich. Optical music recognition with convolutional sequence-to-sequence models. In 18th International Society for Music Information Retrieval Conference, pages 731–737, 201

[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In International Conference on Learning Representations, San Diego, California, USA

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin.2017.Attention Is All You Need.In 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.
[11] Y. Lecun,L. Bottou,Y. Bengio,P. Haffner.1995.Gradient-based learning applied to document recognition.In Proceedings of the IEEE 86(11):2278 – 2324
[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun.2016.Deep Residual Learning for Image Recognition.2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)
[13] Minh-Thang Luong, Hieu Pham, Christopher D. Manning.2015.Effective Approaches to Attention-based Neural Machine Translation.In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing
[14] Ilya Sutskever, Oriol Vinyals, Quoc V. Le.2014.Sequence to Sequence Learning with Neural Networks.In Neural Information Processing Systems (NIPS 2014)
[15] Sepp Hochreiter,Jurgen Schmidhuber.1997.Long Short-Term Memory.Neural Computation 9(8):1735-1780, 1997
[16] Sergey Ioffe, Christian Szegedy.2015.Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.In ICML'15: Proceedings of the 32nd International Conference on International Conference on Machine Learning pages 448-456
[17] Razvan Pascanu, Tomas Mikolov, Yoshua Bengio.2013.On the difficulty of training Recurrent Neural Networks.In ICML'13: Proceedings of the 30th International Conference on International Conference on Machine Learning
[18] Karen Simonyan, Andrew Zisserman.2014.Very Deep Convolutional Networks for Large-Scale Image Recognition.In International Conference on Learning Representations(ICLR) 2015
[19] Kunihiko Fukushima.1980.Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. In Biological Cybernetics volume 36, pages193–202 (1980)
[20] Vinod Nair, Geoffrey E. Hinton.2010.Rectified Linear Units Improve Restricted Boltzmann Machines. In ICML'10: Proceedings of the 27th International Conference on International Conference on Machine Learning