Richard Rivaldo
13519185 – K04

Text Recognition

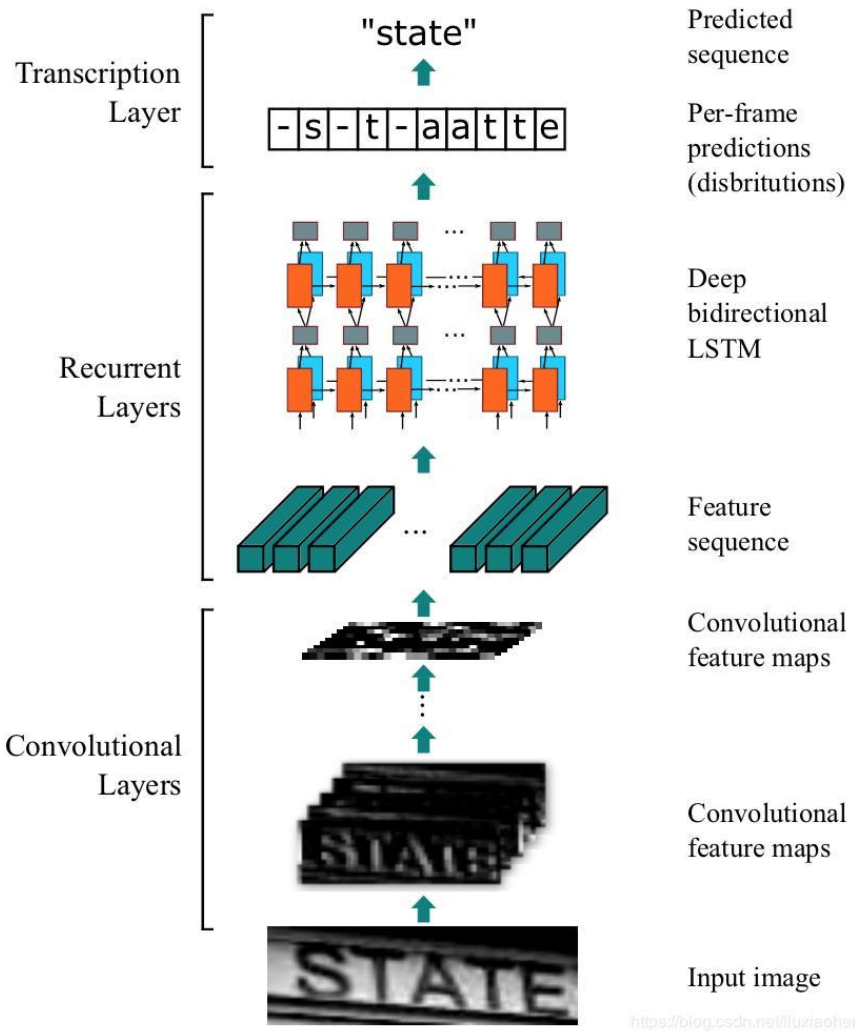
1. Jelaskan arsitektur yang anda buat untuk melakukan ekstraksi teks pada gambar!

   Pertama, gambar yang menjadi input dari arsitektur yang dibuat akan dipreproses terlebih dahulu. Awalnya, akan dilakukan beberapa tahap preproses seperti *deskewing* terhadap gambar supaya orientasi gambar bisa lurus, *resizing* ukuran gambar menjadi ukuran yang uniform untuk semua gambar, konversi *channel* gambar menjadi *grayscaled*, binarisasi gambar dengan menggunakan teknik Otsu, serta penghilangan *noise* melalui transformasi morfologi gambar, erosi dan dilatasi gambar.
   Namun, hasil preproses memberikan hasil yang tidak merata untuk semua gambar, dalam artian ada gambar yang berhasil diproses menjadi lebih baik lagi, tapi ada juga gambar yang teksnya makin tidak terlihat (sebagian besar dikarenakan teks yang ukuran *font* nya terlalu tipis, atau teks yang terlalu berdempetan). Oleh karena itu, pada akhirnya fase preproses hanya dilakukan sampai konversi gambar menjadi *grayscaled*. Semua tahap preproses dilakukan dengan menggunakan bantuan pustaka OpenCV dan *Numpy*. Nantinya, gambar juga akan dinormalisasi dengan membagi elemen gambar dengan 255.
   Adapun arsitektur yang digunakan untuk melakukan *text extraction* adalah arsitektur CRNN, R-CNN, ataupun akronim lainnya. CRNN atau *Convolutional Recurrent Neural Network* adalah kombinasi antara dua buah arsitektur dasar *Deep Learning*, yaitu CNN (*Convolutional Neural Network*) dan RNN (*Recurrent Neural Network*). Adapun detail implementasi dari gabungan keduanya adalah sebagai berikut.
   Arsitektur pertama yang akan menerima input tentunya adalah arsitektur CNN yang menerima gambar sebagai masukannya. CNN akan memiliki peran yang sangat penting untuk melakukan ekstraksi fitur yang ada di dalam gambar. Fitur ini digunakan oleh arsitektur untuk mengenal pola dan bentuk teks yang berbeda-beda pada data masukan. Dengan demikian, model bisa mengenal teks masukan pada data meskipun dengan morfologi bentuk yang berbeda.
   Pada arsitektur CNN ini, akan diterima gambar masukan yang ukurannya telah homogen untuk semua data, yaitu 128 x 32 *pixel* dengan satu buah *channel* melalui *input layer*. Setelah itu, pada CNN akan dilakukan beberapa kali *downsampling* terhadap gambar. Tujuan dari *downsampling* ini adalah supaya model bisa lebih tepat menangkap fitur yang ada di dalam gambar. *Downsampling* dilakukan dengan menggunakan *convolution 2D layer* serta dengan *pooling* melalui *Max Pooling 2D*. Ukuran kernel yang dipakai dalam implementasinya beragam. Terkadang juga dilakukan *batch normalization* untuk mempercepat fase *training*. Adapun juga fungsi aktivasi yang digunakan pada lapisan konvolusional tersebut adalah ReLU.
   Setelah blok *downsampling* terakhir, maka output dari CNN akan di-*squeeze* terlebih dahulu dengan bantuan *layer Lambda* supaya dimensinya sesuai dengan dimensi yang dibutuhkan oleh RNN. Masukan tersebut kemudian akan diteruskan dan dilanjutkan pemrosesannya oleh RNN.
   Untuk RNN sendiri, akan digunakan arsitektur *bidirectional* LSTM sebagai *layer* pemroses utamanya. Jika CNN berurusan dengan masukan gambar, maka RNN akan lebih berurusan dengan masukan data sekuensial, yaitu berupa data label yang setiap karakternya sudah dikonversi menjadi digit angka yang berkorespondensi dengan karakter tersebut. Hal ini mirip dengan pemrosesan data sekuensial yang dilakukan pada *Poem Generation*.
   RNN sendiri akan memberikan keluaran melalui *Dense layer* yang memiliki 27 buah unit. Angka ini berasal dari jumlah karakter unik yang ada pada daftar semua label (teks) yang ada pada gambar tersebut, yaitu semua alfabet *lowercase*, ditambah satu karakter khusus

untuk karakter *padding* sekuens sebelumnya. Adapun fungsi aktivasi yang digunakan di sini adalah *Softmax Activation Function* yang cocok digunakan karena permasalahan ini seolah-olah mirip dengan mengelompokkan teks yang ada pada gambar ke dalam 27 karakter tadi.

Sampai sini, maka model fungsional untuk *testing* sudah bisa dibentuk dengan masukannya adalah masukan CNN awal tadi, sedangkan keluarannya adalah *dense layer* tadi. Untuk *training*, akan ditambahkan *transcription layer* dalam menghitung nilai *loss* dari fase *training* yang akan dilakukan. Pada permasalahan ini akan digunakan *loss* berupa CTC *Layer*, karena *cross-entropy loss* yang biasanya cocok untuk permasalahan multi-klasifikasi tidak cocok dengan arsitektur dan data yang digunakan.

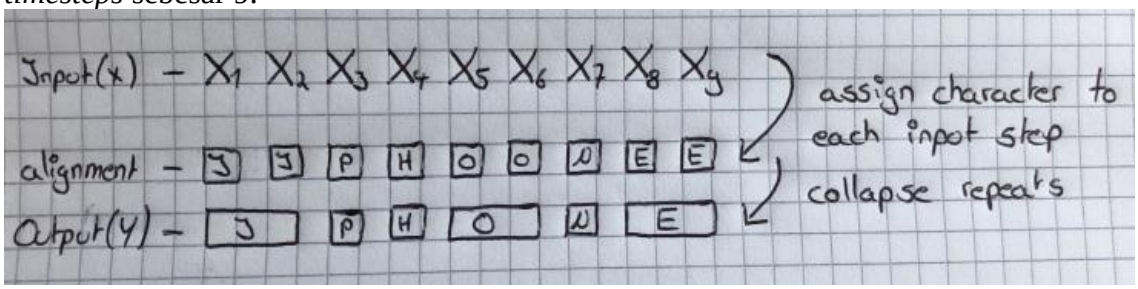
Layer ini akan meneruskan *layer* dari *test model* sebelumnya. Implementasinya sendiri dilakukan dengan menggunakan *Lambda layer* untuk memanggil fungsi CTC *Batch Cost* yang dimiliki oleh *backend* Keras. Selain itu, terdapat beberapa *layer* tambahan yang digunakan sebagai input untuk menghitung CTC ini. Input ini akan dijelaskan lebih lanjut pada pertanyaan poin 2. *Loss function* ini akan digunakan untuk mengevaluasi model melalui validasi terhadap *validation data*, sehingga keluaran dari *training model* ini adalah nilai *loss* tersebut, sedangkan masukannya adalah *input layer* CNN awal dan beberapa *layer* tambahan tadi. Berikut merupakan ilustrasi alur proses arsitektur ini sesuai dengan *paper* berikut.



Richard Rivaldo
13519185 – K04

2. Apa yang kamu ketahui tentang CTC?

Cross-entropy pada permasalahan klasifikasi biasanya merupakan salah satu jenis *loss* yang paling tepat untuk digunakan, selain *Softmax*. Namun, dalam kasus *Text Recognition*, di mana setiap datanya memiliki ukuran panjang kata yang variatif, maka *Cross-entropy* bukan merupakan pilihan yang cocok. Tentunya, bisa dilakukan *alignment* terhadap masukan yang diberikan, namun hal ini akan membutuhkan *resources* yang lebih banyak lagi. Dengan demikian, dibutuhkan sebuah operasi atau cara baru untuk menghitung *loss* dari suatu fase *training* yang dilakukan oleh model.
Operasi ini dikenal dengan CTC atau *Connectionist Temporal Classification*. CTC merupakan operasi yang bisa menangani permasalahan di atas, terutama untuk permasalahan yang membutuhkan *timing* sebagai variabel utamanya. Dalam hal ini, CTC tidak membutuhkan dataset yang *aligned* dalam hal panjang tersebut. Dengan demikian, CTC lebih cocok digunakan untuk permasalahan transkripsi karakter, seperti pada *Speech and Handwritten Recognition* dan *Optical Character Recognition.*
Hal ini dikarenakan CTC bisa melakukan *assignment* probabilitas terhadap label apapun untuk suatu input tertentu dan menjumlahkan semua probabilitas tersebut untuk semua kemungkinan yang ada terhadap *alignment* input dan label keluaran. Untuk mengevaluasi *alignment* yang telah dihasilkannya, maka hasil tersebut akan dibandingkan dengan nilai *ground truth* label untuk input tersebut, sehingga *alignment* dengan *mapping* berdasarkan karakter maupun posisi (*timesteps*) bisa diukur dengan baik kebenarannya. Berikut merupakan contoh *alignment* yang dilakukan oleh CTC untuk jumlah maksimum *timesteps* sebesar 9.



CTC akan melakukan *assignment* label keluaran ke masing-masing *timesteps* dan kemudian menghilangkan karakter yang berulang. Namun, pendekatan ini memiliki permasalahan utama, yaitu jika suatu label memiliki dua buah karakter sama yang berdekatan, maka CTC tidak akan bisa memberikan keluaran yang sesuai.
Oleh karena itu, CTC kemudian akan menggunakan *pseudo-character* yang disebut dengan *blank token*. *Blank token* ini sebenarnya bukan merupakan *whitespace* biasa, tetapi merupakan karakter spesial untuk memisahkan dua input karakter sama yang berdekatan. *Blank token* bisa berada di manapun pada *alignment* CTC. Sebagai contoh, misalkan untuk kata `Hello`, maka kemungkinan *alignment* yang dilakukan oleh CTC adalah `-------H-e-l-l-o----` ataupun `Hel-lo`, namun bukan `-Hello-` jika `-` adalah karakter spesial yang dimaksud.
Dalam implementasinya, akan dibutuhkan beberapa input tambahan seperti yang telah dijelaskan sebelumnya untuk CTC selain input gambar awal, yaitu label, *input length* atau daftar *timesteps* untuk setiap instans data, dan juga *labels length* untuk setiap label yang ada. Keempat input ini dibutuhkan oleh CTC untuk memroses gambar yang dimasukkan sesuai dengan label yang ada, dan melakukan *alignment* pada *timesteps* yang digunakan berdasarkan *label length* tersebut.
Selain bisa digunakan sebagai *loss function*, salah satu kelebihan dari CTC adalah CTC juga bisa digunakan sebagai sebuah *decoder*. Untuk semua kemungkinan *alignment* yang

ada (atau biasa disebut dengan *path*), maka algoritma tercepat yang bisa digunakan adalah aproksimasi dengan *best path decoding* yang terdiri dari dua langkah utama. Langkah pertama adalah mengkalkulasi *best path* yang ada dengan mengambil karakter dengan probabilitas kemunculan tertinggi untuk setiap *timesteps*. Setelah itu, *decoder* akan menghilangkan karakter duplikat dan juga *blank token* yang telah digunakan dalam *alignment*. Dengan demikian, tidak ada lagi *blank token* pada keluaran hasil prediksi model nantinya.

3. Apa yang kamu ketahui tentang attention?

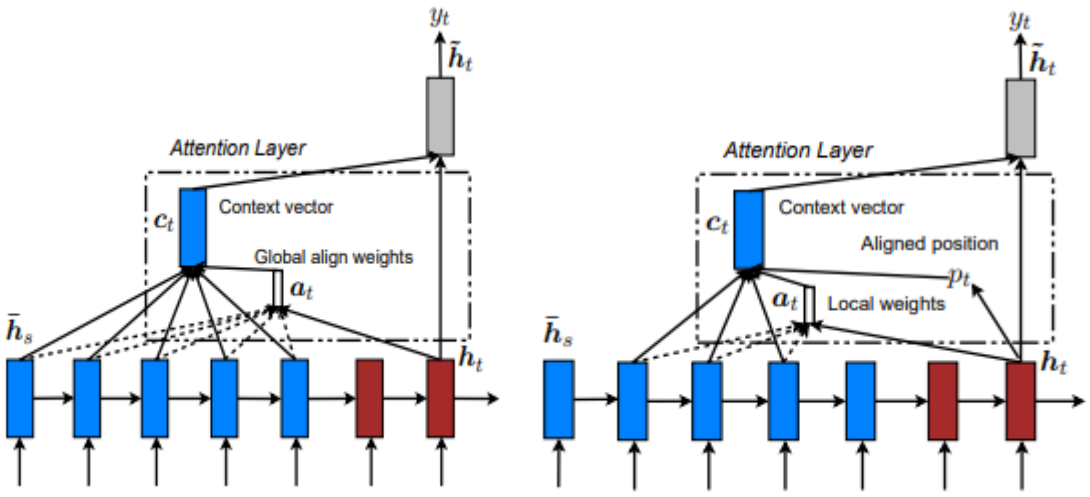
Attention secara garis besar merupakan suatu mekanisme kerja yang mirip dengan bagaimana cara manusia dalam memberikan perhatian lebih kepada tempat atau aspek fitur yang lebih penting dibandingkan fitur-fitur lainnya. Sebagai contoh, ketika kita diberikan suatu gambar anjing dengan *background* lapangan hijau tempat dia bermain, maka kita akan lebih fokus pada anjing sebagai objek utama dalam gambar tersebut.
Di dalam objek tersebut kita pun bisa memberikan tambahan perhatian pada detail-detail yang dianggap menarik dan mencerminkan apa yang ingin disampaikan oleh gambar tersebut. Tidak hanya itu, *attention* tentunya juga bisa diaplikasikan ketika kita melihat suatu teks tertentu. Secara intuitif, kita akan mencoba untuk lebih mengingat bagian-bagian penting dalam teks tersebut seperti subjek, verba, dan objek yang ada di dalam teks tersebut.
Dengan menggunakan prinsip ini, *attention* pada *deep learning* dapat dikatakan sebagai sebuah vektor yang menyatakan bobot ukuran *importance* dari suatu fitur. Bobot ini bisa digunakan sebagai pertimbangan model yang dibangun dalam melakukan inferensi atau prediksi terhadap masukan tertentu. Inferensi ini dilakukan dengan menjumlahkan semua nilai fitur yang ada pada masukan tersebut setelah dibobot dengan vektor tadi.
Tujuan dari mekanisme ini adalah untuk melakukan *break down* terhadap *task* yang kompleks menjadi area-area kecil yang akan diproses secara sekuensial. Hal ini mirip dengan cara pikir manusia yang juga mengandalkan prinsip *Divide and Conquer* ini. Dengan demikian, dari suatu input yang kompleks, maka model bisa fokus pada suatu aspek secara spesifik sampai semua fitur yang ada pada input tersebut berhasil dikategorisasi dengan baik dan tepat.
Mekanisme *attention* seringkali diaplikasikan di dalam *Sequence-to-Sequnece Model* (Seq2Seq) yang biasanya terdiri dari dua komponen utama, yaitu *Encoder* yang berurusan dengan masukan dan *Decoder* yang berurusan dengan keluaran. Pada model Seq2Seq, terdapat *intermediary state* atau yang dikenal juga dengan *hidden state* yang menyimpan *context vector*. *Context vector* ini mirip dengan vektor yang ada pada mekanisme *attention* yang membantu model dalam membobot fiturnya.
Hidden state ini sangat rentan mengalami *bottleneck* dalam memproses informasi, terutama untuk informasi dengan banyaknya poin-poin penting yang harus masuk ke dalam `memori` dari model Seq2Seq, terlebih lagi ketika *context vector* cenderung memiliki ukuran yang tetap. Oleh karena itu, *attention* sangat membantu mekanisme kerja model Seq2Seq dengan mengizinkan *decoder* yang sebelumnya tidak mengetahui *hidden states* yang ada untuk mengakses semua status yang dimiliki model tersebut. Kalkulasi untuk *hidden state* dari model tersebut dapat dilakukan dengan formula berikut.

$$H_k = f(H_{k-1}, Y_{k-1}, C_k)$$

H_k adalah *hidden state* pada waktu *k*, dan H_{k-1} merupakan *hidden state* yang sebelumnya dimiliki oleh *decoder*. Adapun Y_{k-1} adalah output sebelumnya yang dimiliki oleh *decoder*, dan C_k adalah *context vector* yang dimiliki oleh semua *hidden state* dari *encoder*. Adapun mekanisme untuk melakukan komputasi terhadap bobot dari *attention* sendiri dapat dilakukan dengan berbagai macam cara seperti yang ada pada gambar berikut.

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

Model *attention* sendiri bisa diklasifikasikan ke dalam beberapa jenis. Pertama, ada yang disebut dengan *global* dan *local attention*. *Global attention* merupakan model yang sama dengan yang tadi telah dibicarakan. Pada *global attention*, terjadi *global vector alignment* pada *decoder* berdasarkan setiap sumber *hidden states* pada *encoder*. Berbeda dengan *global attention*, maka *local attention* hanya mengambil beberapa sumber *hidden states* dalam melakukan *alignment* tersebut. *Local attention* bisa terdiri dari dua buah *alignment*, yaitu *predictive alignment* dan *monotonic alignment*.



Model *attention* lainnya adalah *hard* dan *soft attention*. *Soft attention* secara sederhana merupakan mekanisme *attention* yang menggunakan rataan dari bobot *attention* semua fitur dan mengaplikasikannya kepada semua fitur yang ada pada data input. Secara tidak langsung, maka fitur-fitur yang tidak penting akan diberikan bobot yang sangat rendah sehingga meskipun semua fitur pada gambar atau teks terbobot, model tetap bisa fokus pada fitur yang penting. Berbeda dengan *soft attention*, *hard attention* hanya fokus pada

suatu fitur spesifik pada data masukan. Untuk mengestimasi gradien semua status yang ada, maka perlu diambil nilai rataan *sampling* dengan menggunakan metode Monte Carlo. Terakhir, ada yang disebut dengan *self-attention* atau *intra-attention*. Seperti namanya, *self-attention* merupakan mekanisme *attention* yang dilakukan oleh suatu masukan terhadap dirinya sendiri. Mekanisme ini akan berusaha untuk mencari kemungkinan posisi dan urutan sekuens yang berbeda dari masukan awal untuk menemukan representasi serupa dari sekuens tersebut. Salah satu arsitektur terkenal yang memanfaatkan mekanisme ini adalah *transformer*.

Referensi Tambahan

An Intuitive Explanation of Connectionist Temporal Classification.
Attention? Attention!
Attention Mechanisms in Recurrent Neural Networks (RNNs) With Keras.
Attention Models.
Brief Introduction to Attention Models.
Connectionist Temporal Classification.
Understanding Attention In Deep Learning.