BAB III

LANDASAN TEORI

3.1 Python

Python merupakan Bahasa pemrograman tingkat tinggi multiguna yang dapat digunakan untuk mengembangkan berbagai macam keperluan perangkat lunak. Python memiliki berbagai macam paradigma, utamanya namun tidak terbatas pada pemrograman berorientasi objek. Kode pada Python dirancang untuk memiliki tingkat keterbacaan yang tinggi (readability). Python juga memiliki *standard library* yang mendukung berbagai macam tugas pemrograman (Lutz, 2013). Ukuran kode Python secara umum hanya 1/3 hingga 1/5 kode yang dibuat dengan Bahasa pemrograman C++ atau Java. Python secara otomatis mengalokasikan memori untuk objek dan menghapusnya ketika objek sudah tidak digunakan.

3.2 Natural Language Processing

Natural Language Processing merupakan cabang dari ilmu komputer yang berkembang dari studi Bahasa dan komputasi linguistik dalam cabang kecerdasan buatan. Tujuan dari NLP adalah untuk mengembangkan dan merancang aplikasi yang mampu menjadi penengah antar interaksi manusia dan mesin dengan perangkat lain melalui penggunaan Bahasa alami (Pustejovsky & Stubbs, 2012). Sebelum perancangan model, diperlukan beberapa tujuan pembuatan model seperti, untuk apa model dibuat, *corpus* atau *dataset* seperti apa yang diperlukan, dan tujuan apa yang perlu dicapai dalam pembuatan model.

Dalam pengumpulan *dataset* diperlukan beberapa teknik *text pre-processing* agar *dataset* yang tidak terstruktur memiliki representasi yang terstruktur. Beberapa proses *pre-processing* adalah:

1. Tokenization

   Merupakan proses pemotongan *string* menjadi teks – teks independen yang memiliki makna. Proses tokenisasi pada suatu teks akan memotong teks tersebut menjadi beberapa kalimat atau kata (Sanger & Feldman, 2007).

2. Part-of-Speech Tagging

   Merupakan proses pemberian POS Tag pada suatu kalimat atau kata. *POS Tagging* dapat dilakukan dengan memberi tiap kata *tag* seperti kata kerja, kata benda, kata sifat dan aturan grammar lain (Sanger & Feldman, 2007).

3. Syntactical Parsing

   Merupakan proses analisis sintaks suatu kalimat berdasarkan teori grammar tertentu. Proses *parsing* yang umum dapat dibagi dua yaitu *dependency* dan *constituency* (Sanger & Feldman, 2007).

3.3 Struktur Kalimat

Kalimat adalah satuan Bahasa terkecil dalam wujud lisan atau tulisan, yang mengungkapkan pikiran yang utuh (Alwi, Lapoliwa, & Darmowidjojo, 2007). Kalimat yang umum digunakan dalam Bahasa Indonesia adalah kalimat Simpleks. Kalimat simpleks merupakan kalimat yang terdiri atas satu klausa atau satu struktur predikat seperti kalimat yang berpola S-P-O-K. Struktur inti kalimat Bahasa Indonesia yang paling sederhana terdiri dari fungsi subjek dan predikat. Struktur kalimat dasar bahasa Indonesia dapat berupa S-P, S-P-O, S-P-Pelengkap, S-P-O-Pelengkap, S-P-O-K, dan S-P-K. Kalimat yang memiliki struktur umum tersebut dapat diperluas seperti contoh: (1) S-P-Pel menjadi (1a)

S-P-Pel-K, (1b) K-S-P-Pel. Dengan mengikuti struktur kalimat dasar, kalimat yang baik dapat dibentuk (Indonesia, 2015).

3.4 Machine Learning

Machine Learning merupakan subset dari studi kecerdasan buatan yang berfokus untuk membuat suatu komputer bisa mempelajari pola dari suatu data. Pembuatan algoritma atau model statistik yang mampu mempelajari dan membuat prediksi pada data tertentu merupakan tugas utama dalam *machine learning*. Algoritma *machine learning* umumnya mampu pelajari pola atau tren tertentu pada suatu data dan dapat memberi pengetahuan yang tidak ekplisit pada pengguna mengenai data. Contoh penerapan *machine learning* seperti deteksi penipuan pada transaksi kartu kredit dan sistem yang memfilter informasi mengenai preferensi baca pengguna (Mitchell, 1997). Salah satu tugas utama dalam *machine learning* adalah menentukan algoritma mana yang tepat digunakan untuk menyelesaikan tugas komputasi tertentu.

Dalam *machine learning*, ada 2 tugas yang umum dilakukan yaitu:

1. *Supervised Learning*

   Supervised learning adalah metode pembelajaran dengan cara memberi model suatu contoh input dan pasangan kelas output dari input tersebut. Model akan mempelajari aturan atau relasi umum yang memetakan input ke output. Beberapa metode *supervised learning* yang dapat digunakan dalam kondisi tertentu adalah *Semi-supervised learning, Active learning, dan Reinforcement learning.*

2. *Unsupervised Learning*

   Unsupervised learning adalah metode pembelajaran tanpa memberi label pada sampel input. Metode ini memiliki tujuan untuk menemukan pola implisit dari suatu data.

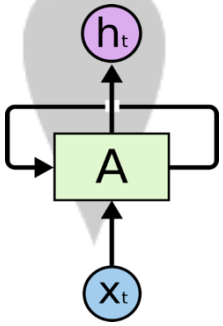
3.5 Deep Learning

Deep Learning atau *deep hierarchical learning* merupakan bagian dari metode *machine learning* yang tidak terbatas pada tugas spesifik tertentu. Model *deep learning* umumnya memiliki lapisan (*layer)* yang berjumlah lebih dari satu. Informasi dialirkan melalui fungsi non-linier dalam lapisan model untuk mengekstraksi fitur maupun pola dari data (Deng & Yu, 2014). *Deep learning* berkaitan dengan mempelajari representasi *multi-level* dari data citra, suara maupun teks. Tiga aspek yang mempengaruhi kepopuleran *deep learning* adalah meningkatnya kemampuan pemrosesan chip yang beredar, kemajuan dalam riset *machine learning* dan riset pemrosesan sinyal atau informasi.

Secara kasar, *Deep Learning* mencoba mengimitasi cara kerja pemrosesan informasi dan pola komunikasi sistem syaraf biologis. Dua masalah utama dalam *deep learning* adalah *overfitting* karena model yang terlalu kompleks dan waktu komputasi. *Overfitting* pada *deep learning* umumnya diatasi dengan berbagai macam teknik regularisasi seperti *dropout.* Untuk menyelesaikan masalah komputasi yang lama, proses komputasi model *deep learning* dilakukan secara paralel dengan GPU (*Graphics Processing Unit).*

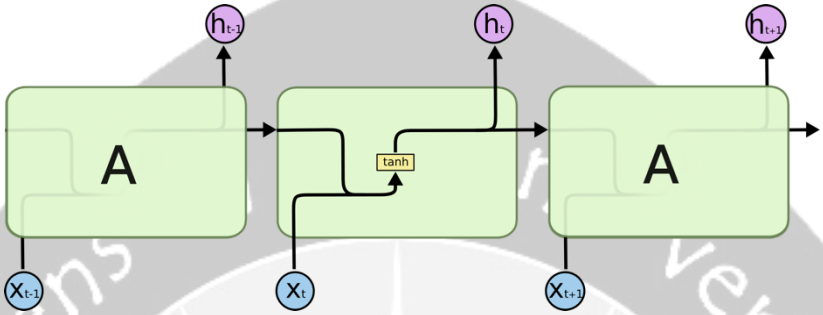
3.6 Recurrent Neural Network

Recurrent Neural Network adalah salah satu jenis jaringan syaraf tiruan yang memiliki koneksi berulang pada *cell* jaringanya. Koneksi berulang ini memungkinkan RNN untuk memproses deretan input.



Gambar 3. 1 *Recurrent Neural Network (Olah,2015)*

Dalam beberapa tahun kebelakang RNN berhasil meraih hasil yang baik dalam menyelesaikan tugas NLP. Dalam *cell* RNN terdapat satu *layer* fungsi tan hiperbolik.

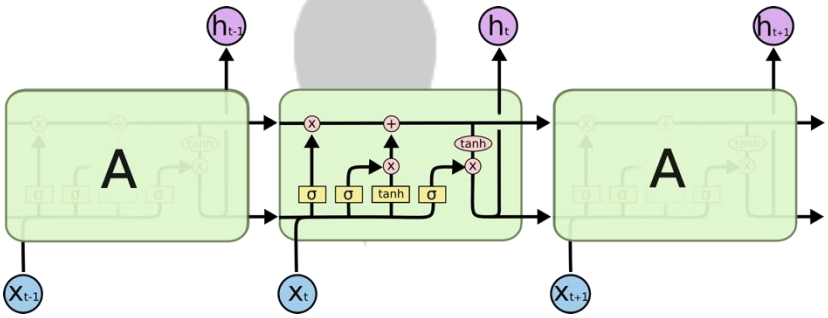


Gambar 3. 2 *RNN Dijabarkan terhadap Timestep. (Olah, 2015)*

Hidden state dalam RNN ini berfungsi seperti memori untuk memproses deret input.

3.7 Long Short-Term Memory Network

Long Short-Term Memory Network adalah salah satu jenis jaringan syaraf tiruan yang merupakan varian dari *Recurrent Neural Network*. LSTM dapat mempelajari ketergantungan jangka panjang (Long-term Dependencies) yang sebelumnya menjadi kelemahan dalam RNN. LSTM juga memiliki koneksi berulang atau struktur yang seperti rantai. Perbedaan LSTM dan RNN terletak pada lapisan yang terdapat dalam setiap *cell* LSTM.

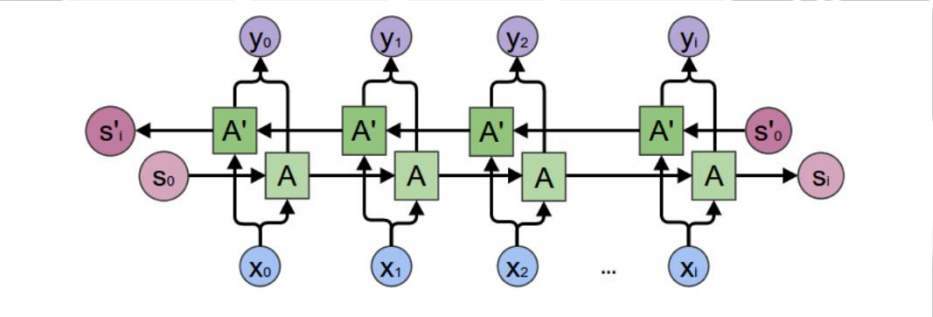


Gambar 3. 3 *Cell LSTM dengan 4 Lapis Fungsi Aktivasi (Olah, 2015)*

Dalam setiap *cell* LSTM terdapat 3 fungsi sigmoid dan 1 fungsi tan hiperbolik. Untuk permasalahan ketergantungan jangka panjang, LSTM dapat mengatasi *noise*, representasi terdistribusi, dan nilai kontinu (Hochreiter & Schmidhuber, 1997).

3.8 Bidirectional LSTM

Bidirectional LSTM adalah salah satu varian LSTM yang umum digunakan. Input yang dimasukkan ke dalam BiLSTM ada 2 jenis yaitu input *forward* dan input *backward.* Output dari lapisan ini umumnya digabungkan menjadi satu. Dengan *layer* ini, model dapat mempelajari informasi masa lalu (*past)* dan informasi masa mendatang (*future)* untuk tiap sekuen input.



Gambar 3. 4 Arsitektur BiLSTM. *(Olah, 2015)*

3.9 Aktivasi ReLU

Dalam *machine learning* ReLU atau *Rectifier Linear Unit* adalah fungsi aktivasi yang memiliki persamaan:

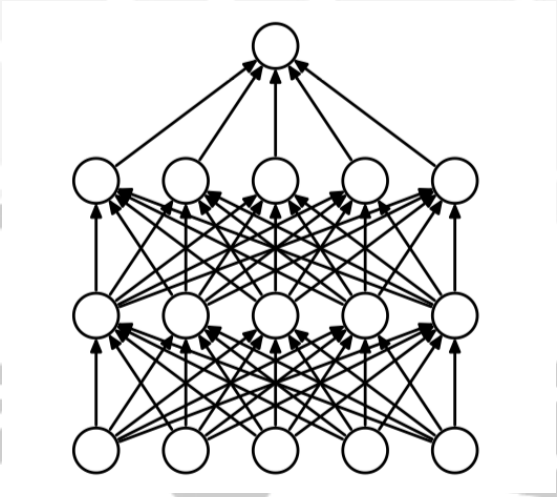
$$f(x) = max(0, x)$$

Persamaan 3. 1 *Fungsi Aktivasi ReLU*

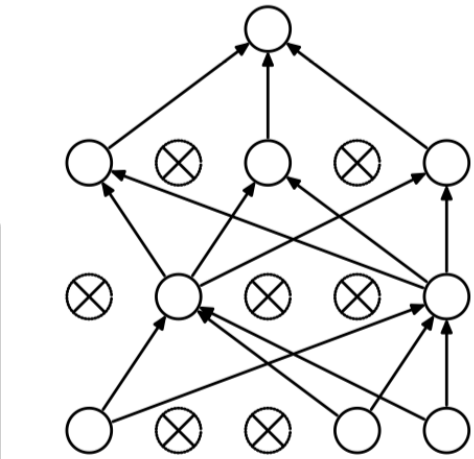
Pemilihan fungsi aktivasi dalam jaringan syaraf tiruan merupakan hal penting. ReLU merupakan fungsi aktivasi yang paling popular dan banyak diterapkan dalam pembangunan model *neural network* (Ramachandran, Zoph, & Le, 2017). Range dari fungsi aktivasi ReLU adalah [0, ∞]. Semua input nilai negatif pada fungsi aktivasi ReLU akan menjadi nilai 0.

3.10 Dropout

Dropout merupakan teknik regularisasi model jaringan syaraf tiruan untuk mengurangi *overfitting* pada dataset. *Dropout* merupakan cara yang efisien untuk menghindari *overfitting. Deep neural network* umumnya memiliki lapisan yang banyak yang memungkinkan model mempelajari relasi kompleks antara input dan output. Dengan jumlah dataset yang terbatas, *Deep neural network* mungkin justru akan mempelajari relasi kompleks *noise* dari dataset (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). *Dropout* dilakukan dengan cara mematikan (*dropping out)* unit dari lapisan tersembunyi (*hidden layer)* maupun lapisan yang tampak (*visible layer).*



Gambar 3. 5 *Neural Network Sebelum Diaplikasikan Dropout*



Gambar 3. 6 *Neural Network Setelah Diaplikasikan Dropout.*

3.11 ADAM Optimizer

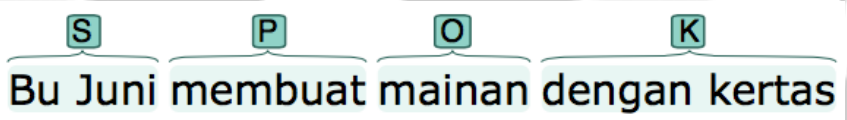
ADAM adalah algoritma yang digunakan untuk optimisasi *gradient* pada *neural network* berdasarkan data latih. Metode ini lebih mudah diimplementasikan, efisien secara komputasi, memerlukan kebutuhan memori yang kecil, dan sesuai untuk masalah yang memiliki banyak data maupun parameter (Kingma & Ba, 2015). ADAM berbeda dengan algoritma *Stochastic Gradient Descent* yang memiliki laju pembelajaran (*learning rate)* yang tidak berubah setiap pembaruan bobot (*weights update).* Algoritma ADAM menghitung rata – rata bergerak eksponensial (*Exponential Moving Average)* dari gradien dan gradien kuadrat. Parameter beta1 dan beta2 mengendalikan laju pengurangan (*decay rate)* dari rata – rata bergerak. (Brownlee, 2017)

BAB IV

PERSIAPAN DATA DAN MODEL

4.1 Persiapan Data

Pada penelitian ini terdapat dataset yang terdiri dari kalimat yang memiliki struktur kalimat dasar Bahasa Indonesia. Setiap kalimat pada dataset ditandai dengan tag S yang berarti subjek, tag P yang berarti Predikat, tag O yang berarti Objek, tag Pel yang berarti Pelengkap, dan tag K yang berarti Keterangan. Pemberian tag S-P-O-K pada tiap kata maupun kalimat mengikuti standar IOB *tagging*. IOB adalah format penanda yang umum dalam berbagai tugas pemrosesan bahasa alami.



Gambar 4. 1 *Contoh IOB Tagging*

Pada contoh Gambar 4.1, Bu Juni memiliki tag S atau Subjek. Dalam *file* dataset Bu Juni memiliki tag seperti: Bu B-S\nJuni I-S\n. Pada akhir kata atau token diberikan karakter *end-of-line.* Pada tiap akhir kalimat diberi karakter *end-of-line* ganda "\n\n". Di antara token/kata dan tag diberi *whitespace*. Tag B-S dalam kata bu berarti permulaan kata dari subjek. B memiliki arti *Begin* atau permulaan. Tag I-S dalam kata Juni berarti bagian dari frase Subjek. I memiliki arti *Inside* atau di dalam kelompok kata. Kata atau kelompok kata yang tidak memiliki tag SPOK akan mendapat tag O yang berarti *outside* atau bukan merupakan kelompok kata manapun.

Dataset berjumlah 3627 kalimat. Pembagian data latih 70% dari dataset. Pembagian data validasi 30% dari dataset.

***Tabel 4. 1** Contoh IOB Tagging*

Kalimat	Bu	Juni	membuat	mainan	dengan	kertas	.
Struktur	Subjek		Predikat	Objek	Keterangan		
IOB Tag	B-S	I-S	B-P	B-O	B-K	I-K	O

Pembagian dengan menggunakan data tes memiliki rasio yang berbeda yaitu 70% data latih, 20% data validasi, 10% data tes. Sebelum dilakukan pembagian, dataset diacak terlebih dahulu untuk menghindari pengambilan sampel hanya dari salah satu distribusi.

4.2 Alat Penelitian

Alat penelitian yang digunakan dalam penelitian ini terdiri dari 2 komponen yaitu perangkat keras dan perangkat lunak. Perangkat keras dan perangkat lunak yang digunakan adalah sebagai berikut:

1. Perangkat Lunak

Perangkat lunak yang digunakan adalah sebagai berikut:

a. Nama : Ubuntu 16.04 LTS Desktop
   Sumber : Canonical Ltd.
   Ubuntu adalah sistem operasi yang digunakan untuk penelitian, pengkodean dan pengujian model.

b. Nama : Visual Studio Code
   Sumber : Microsoft Corp.

Visual Studio Code adalah *source code* editor untuk mengembangkan model. Editor ini juga digunakan untuk melakukan *debugging* pada saat pembangunan model.

c. Nama : Tensorflow 1.8
Sumber : Google Brain Team
Tensorflow adalah *library open-source* yang dikembangkan google untuk keperluan produksi dan riset *machine learning*. Tensorflow juga memiliki *support* komputasi dengan GPU berbasis CUDA. Dalam penelitian ini tensorflow digunakan sebagai *backend* dari API tingkat tinggi *Keras*.

d. Nama : Keras 2.2.0
Sumber : Francois Chollet
Keras adalah API tingkat tinggi *deep learning* dalam Bahasa pemrograman Python. Keras dapat berjalan dengan menggunakan *backend* Theano maupun Tensorflow.

e. Nama : CUDA 9.0
Sumber : NVIDIA Corp
CUDA merupakan platform dan API pemrograman paralel dengan kartu grafis NVIDIA.

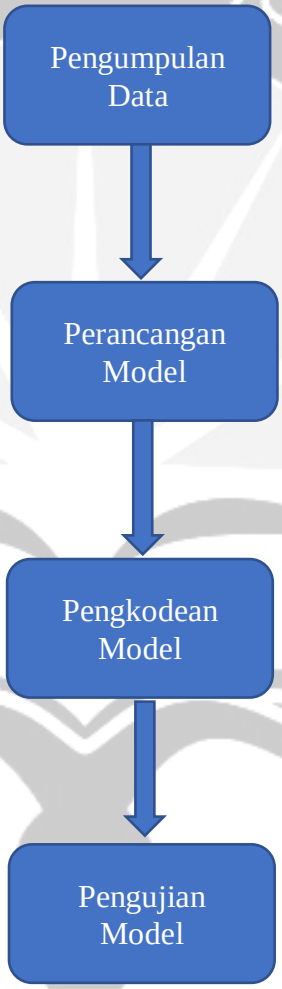
2. **Perangkat Keras**

Perangkat keras yang digunakan dalam penelitian ini adalah sebagai berikut:

a. PC (Personal Computer) dengan spesifikasi:
- CPU : Intel Core i7-6700K
- GPU : NVIDIA GeForce GTX 1060, Arsitektur Pascal, Memori Global 6GB GDDR5X, CUDA Cores 1280, Memory Bus 192-Bit, Memory Bandwidth 192 GB/s
- RAM : 8192 MB

4.3 Langkah-Langkah Persiapan Data dan Model

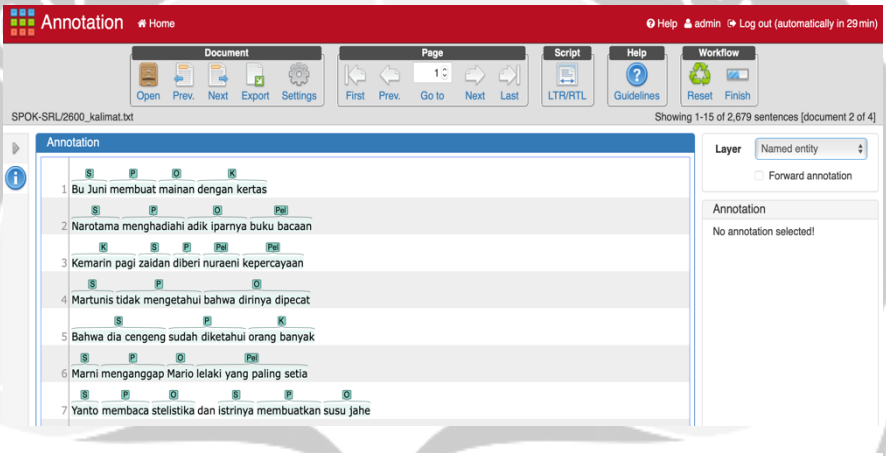
Tahap – tahap yang dilakukan dalam proses persiapan data dan model adalah pengumpulan data, perancangan model, pengkodean model dan pengujian model. Di bawah ini adalah bagan yang menunjukkan tahap persiapan data dan model:



Gambar 4. 2 *Diagram Persiapan Data dan Model.*

4.3.1 Pengumpulan Data

Data yang dikumpulkan digunakan untuk melatih model NLP. Data terdiri dari kalimat Bahasa Indonesia yang memiliki struktur atau pola S-P-O-K. Kalimat dengan pola paling sederhana yang dikumpulkan adalah kalimat berpola S-P. Kalimat dikumpulkan dari beberapa surat kabar Indonesia seperti majalah historia, Kompas, dan Beritagar. Data yang terkumpul kemudian diberi label (anotasi) yang sesuai dengan *tag* struktur kalimat. Anotasi dilakukan secara manual dengan mengacu buku Tata Bahasa Baku Bahasa Indonesia.



Gambar 4. 3 *Proses Anotasi dengan WebAnno.*

Data yang telah dianotasi dieksport dengan format "*kalimat* ||| *tag*" dalam file dengan ekstensi .txt. Kesalahan dalam pengetikan label

```
2 Perayaan Imlek juga diadakan di kampung-kampung |||  B-S I-S B-P I-P B-K I-K
2 Kejadian itu berlangsung begitu cepat sehingga Novel tak sempat mengelak |||  B-S I-S B-P B-Pel I-Pel B-K I-K I-K I-K I-K
1 Novel langsung dibawa ke Rumah Sakit Mitra Keluarga Kelapa Gading , Jakarta Utara |||  B-S B-P I-P B-K I-K I-K I-K I-K I-K I-K I-K I-K I-K I-K
3 Polsek Kelapa Gading kemudian melakukan olah tempat kejadian perkara |||  B-S I-S I-S B-P I-P B-O I-O I-O I-O
```

Gambar 4. 4 *Contoh Dataset Sesuai Format Penelitian.*

diperbaiki secara manual. Dataset yang sudah terkumpul dibagi dengan proporsi 70% data *train*, 20% data validasi, dan 10% data tes. Sebelum pembagian data, data diacak agar masing – masing proporsi data memiliki distribusi kalimat yang merata.

Struktur kalimat yang terdapat di dalam dataset digambarkan dalam tabel 4.2.

Tabel 4. 2 *Jumlah kalimat dengan pola tertentu pada dataset.*

Pola	**Jumlah**
S-P	603
S-P-O	604
S-P-Pelengkap	605
S-P-K	605
S-P-O-K	605
S-P-Pelengkap-K	605

Berdasarkan tabel 4.2, seluruh pola kalimat memiliki jumlah yang hampir sama pada dataset. Jumlah masing – masing pola kalimat yang hampir sama membuat dataset memiliki data yang seimbang.

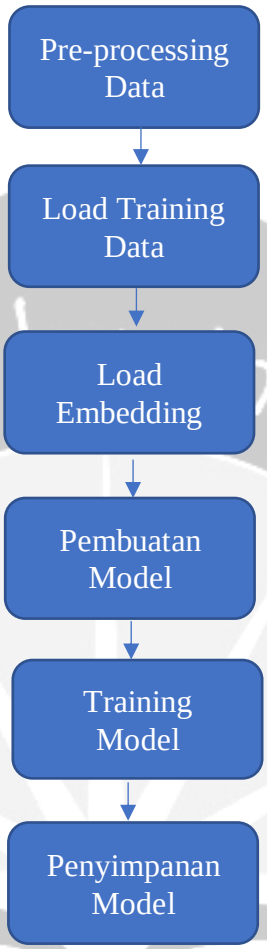
4.3.2 Perancangan Model

Dalam proses perancangan model perlu dibuat suatu alur kerja program. Alur kerja dalam perancangan model ada 2 yaitu alur untuk pelatihan model dan alur untuk prediksi suatu kalimat. Alur untuk pelatihan model identifikasi struktur kalimat ini adalah tahap *pre-processing data*, tahap *Load training data*, tahap *Load Embedding/word vector*, tahap pembuatan model, tahap training model, dan tahap penyimpanan model. Setiap input yang akan masuk ke dalam model akan melewati alur kerja tersebut. Alur untuk prediksi suatu kalimat terdiri dari tahap *Load model* dan tahap prediksi struktur kalimat.

Tahap *pre-processing* data pada alur kerja pelatihan model berfungsi untuk mengolah data menjadi input model yang dibuat. Proses ini diawali

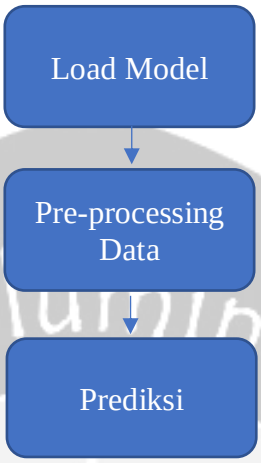
dengan tokenisasi teks atau pemotongan kalimat menjadi sekumpulan kata. Proses selanjutnya adalah *one-hot encoding* untuk mengubah setiap token tadi menjadi representasi integer. Proses selanjutnya adalah menambahkan *padding* angka 0 pada akhir kalimat karena panjang setiap input kalimat bisa berbeda – beda. Pada tahap *training data*, input yang sudah melalui *pre-processing* akan dimasukkan ke dalam model. Data yang telah melalui *pre-processing* akan terbagi menjadi dua yaitu x_train dan y_train. X_train merupakan *list* atau *array* yang elemennya berisi input kalimat. Y_train merupakan *list* atau *array* yang elemennya berisi label dari masing – masing input. *List* x_train dan y_train akan dijadikan input untuk model yang dibuat dengan *API Keras.* Pada tahap *Load Embedding*, program akan menginisialisasi *embedding* atau *word vector* yang ada. Dalam penelitian ini, *embedding* atau *word vector* merupakan model *embedding pre-trained* milik *fasttext* (Grave, Bojanowski, Joulin, & Mikolov, 2016). *Embedding* yang sudah dimuat akan digunakan sebagai *layer* kedua setelah *layer input* pada model.

Tahap pembuatan model adalah tahap penyusunan arsitektur model yang dibuat dengan memilih jenis dan *hyperparameter layer* pada tiap lapisan. Lapisan model yang sudah disusun akan digunakan dalam proses pelatihan dan prediksi struktur kalimat. Dalam tahap *training data*, model akan mempelajari input yang dimasukkan. Proses pelatihan model bergantung berapa lama *epoch* yang dimasukkan peneliti. Dalam proses pelatihan, peneliti juga bisa mengubah *parameter* latihan yang mungkin akan mempengaruhi kemampuan model seperti, *batch size*, jenis *optimizer*, *loss function*, dan metrik yang digunakan dalam mengukur akurasi. Proses penyimpanan model terjadi secara bersamaan atau pada akhir proses training. Pada setiap akhir *epoch*, model akan disimpan berdasarkan epoch yang akan digunakan dalam penelitian sehingga yang tersimpan hanya model pada *epoch* tertentu saja.



Gambar 4. 5 *Alur Kerja Proses Pelatihan.*

Pada alur kerja prediksi, terdapat 3 tahap yaitu *load model*, *pre-processing data*, dan prediksi. Tahap *load model* merupakan tahap memuat model yang sudah melalui proses pelatihan sebelumnya. Model yang dimuat akan digunakan untuk melakukan prediksi pada input kalimat. Komponen – komponen model yang dimuat meliputi arsitektur model, *hyperparameter* model, dan bobot model. Tahap *pre-processing* ini sama dengan tahap yang ada pada proses alur kerja pelatihan. Kalimat yang akan diprediksi harus melewati tahap *pre-processing* agar bentuk input sesuai dengan yang diharapkan model. Tahap terakhir dari alur kerja ini adalah tahap prediksi. Pada tahap prediksi, input yang dimasukkan ke dalam model akan menghasilkan output *list* yang memiliki elemen hasil prediksi.



***Gambar 4. 6** Alur Kerja Proses Prediksi.*

Kedua alur kerja tersebut merupakan gambaran umum perancangan model identifikasi struktur kalimat Bahasa Indonesia. Kedua alur kerja dirancang agar identifikasi struktur kalimat berjalan dengan optimal.

4.3.2.1 Pre-processing Data

Preprocessing merupakan salah satu tahap terpenting agar model berjalan dengan optimal. Proses pertama adalah tokenisasi. Pada saat tokenisasi, kalimat yang akan input dipotong menjadi kata. *Tokenizer* yang digunakan dalam penelitian ini adalah *tokenizer* NLTK. *Tokenizer* juga akan memisahkan kata dari simbol seperti tanda tanya, tanda seru, titik, koma dan simbol lain. Setelah tokenisasi, proses selanjutnya adalah *one-hot encoding*. Proses *One-hot encoding* akan mengubah token menjadi representasi integer. Untuk mengubah token menjadi representasi integer, diperlukan *word dictionary*.

```
def tokenize(str_sequence):
  """ Fungsi untuk Tokenisasi suatu kalimat"
    Input format: String
    Return:
      list kalimat: [[words]]
  """

  sentences = []
  words = word_tokenize(str_sequence)
  sentences.append((words))
  return sentences
def string_sequence_to_ids_dictwise(str_seq,
dictionary, lowercase=False,
pretrained_embeddings=None):
  """token/kata yang tidak terdapat dalam wordvec
akan dianggap sebagai UNKNOWN_TOKEN
  """
  ids = []
  for s in str_seq:
    if s is None:
      ids.append(-1)
      continue
    if lowercase:
      s = s.lower()
    if (pretrained_embeddings != None) and not (s
in dictionary.str2idx) :
      s = UNKNOWN_TOKEN
    ids.append(dictionary.get_index(s))
  return ids
#Pad input sequnces with 0
    x_train = pad_sequences(train_token,
maxlen=MAX_SEQUENCE_LENGTH,padding='post')
```

***Gambar 4. 7** Kode Preprocessing Input.*

Word dictionary adalah kumpulan kata yang akan digunakan dalam proses mengubah token menjadi integer. Sebagai contoh, jika ada token/kata "saya" dalam input kalimat, program akan mencari kata "saya" di *word dictionary*. Kemudian program akan mengecek ada di indeks keberapa kata yang dicari tersebut. Jika kata yang dicari berada di indeks 1 maka token "saya" akan memiliki representasi integer 1. Indeks *one-hot encoding* merupakan

angka yang dipilih manasuka untuk tiap kata. Tiap kata harus memiliki representasi integer yang unik. Jika angka 1 sudah digunakan untuk merepresentasikan kata/token "saya" maka angka 1 tidak bisa digunakan untuk kata/token lain. Dalam penelitian ini, indeks 0 dipakai untuk representasi token "*Masking*" karena indeks 0 akan diabaikan pada *layer embedding*. Indeks 1 dipakai untuk token "*UNKNOWN*". Token "*UNKNOWN*" dipakai untuk merepresentasikan token/kata yang tidak terdapat di dalam *word dictionary*.

Setelah proses *one-hot encoding*, *list* yang sudah dalam representasi integer akan diberi *padding* pada bagian akhir dengan angka 0. Karena model hanya bisa menerima input yang jumlahnya tetap sedangkan panjang input kalimat bervariasi, maka input kalimat harus diberi *padding*. Panjang tetap dalam penelitian ini ditetapkan sebanyak 100 token. Sebagai contoh, semisal terdapat input dengan panjang 50 token, maka input tersebut akan diberi *padding 0* sebanyak 50. Token terpanjang yang ada dalam dataset adalah kalimat dengan 46 token. Untuk menghindari terpotongnya input karena kalimat memiliki token lebih banyak daripada *neuron* input, maka panjang tetap ditetapkan 100 token.

4.3.2.2 Load Training Data

Data hasil *preprocessing* adalah *array* yang memiliki dimensi (n,100). Dataset terbagi menjadi dua yaitu data *train* dan data validasi. Data *train* tersimpan dalam *array* x_train dan data validasi tersimpan dalam *array* x_val. Label data *training* tersimpan pada *array* y_train dan label data validasi tersimpan pada *array* y_val. Sebelum label dimasukkan ke dalam model, label juga diubah menjadi *one-hot vector*. Sebagai contoh, label B-S yang pada *label dictionary* memiliki index 1 akan diubah menjadi "0 1 0 0 0 0 0 0 0 0 0" dan label O yang memiliki index 11 akan diubah menjadi "0 0 0 0 0 0 0 0 0 0 1". Setelah proses perubahan ke *one-hot vector*, label memiliki dimensi (n,11) dimana n adalah jumlah data.

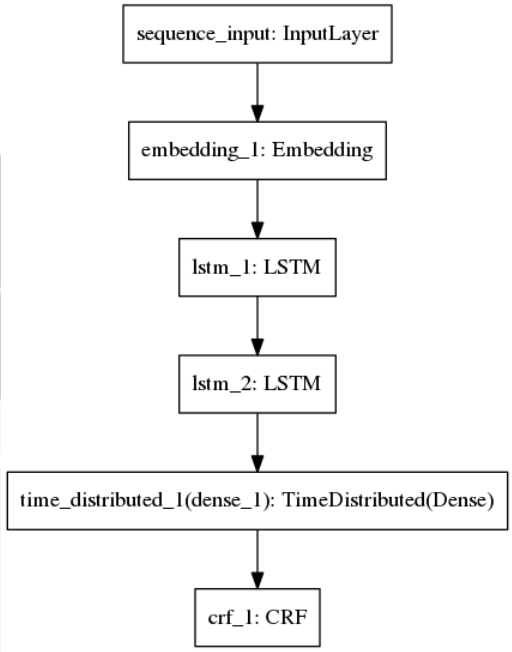
4.3.2.3 Load Embedding

Untuk tahap *load embedding*, *embedding* yang berukuran (300000,300) akan disimpan dalam objek atau *python dictionary*. Token "*UNKNOWN*" akan dimasukkan kedalam *embedding* dengan koefisien yang diisi dengan angka acak. Sebelum digunakan dalam *embedding layer*, *list* akan diinisialisasi dan diisi dengan token yang ada pada dataset dan nilai koefisien yang sesuai dengan nilai yang ada pada *embedding*. Kemudian, *list* inilah yang akan digunakan pada parameter bobot *layer embedding*.

Parameter yang ada pada *embedding layer* adalah, jumlah kata, dimensi *embedding*, panjang input, *trainable*, dan *mask zero*. Jumlah kata akan berjumlah sama dengan *word dictionary* kecuali ada kata di *word dictionary* yang tidak terdapat pada *word embedding*. Dimensi *pre-trained embedding* milik *fasttext* adalah 300. Panjang input seperti kesepakatan dalam tahap preprocessing adalah 100 token. Parameter *trainable* harus diset *false* agar ketika proses *training* bobot dari *embedding* tidak berubah. Parameter *mask_zero* digunakan untuk mengabaikan indeks 0 karena angka 0 dalam model ini digunakan sebagai *padding*.

4.3.2.3 Pembuatan model

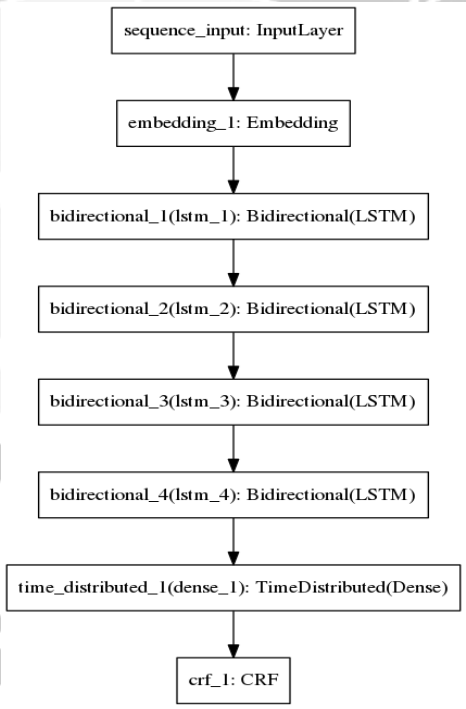
Pada tahap ini model akan didefinisikan dengan kode. Tahap paling awal adalah definisi *layer input*. *Layer input* memiliki parameter *shape* untuk mengatur bentuk input yang masuk ke dalam model, dtype untuk mendefinisikan tipe data input, dan nama *layer*. Karena setiap input memiliki panjang tetap 100, maka parameter *shape* akan didefinisi dengan angka 100 dan tipe data int32. *Layer* selanjutnya adalah *layer embedding* yang sudah dinisialiasi sebelum pembuatan model. *Layer embedding* menerima masukan dari *layer input*.



Gambar 4. 8 *Graph Model LSTM*

Setelah *layer input* dan *embedding*, *layer* LSTM atau Bi-LSTM akan didefinisikan. Tiap *layer LSTM* memiliki *hyperparameter* yang harus ditentukan. Pada *layer* LSTM, beberapa *hyperparameter* yang penting adalah jumlah unit *cell*, kondisi *forget bias* pada tiap unit, *dropout layer* LSTM, dan *recurrent dropout* pada unit LSTM. Setelah *layer* LSTM, *layer dense* atau *fully connected layer* akan didefinisikan. *Layer dense* memiliki parameter jumlah unit *cell* yang akan dipakai dan fungsi aktivasi yang akan digunakan. Dalam penelitian ini, *dense layer* menggunakan fungsi aktivasi ReLU. Lapisan *fully connected* juga akan diberi *classwrapper* bawaan *Keras* yaitu *TimeDistributed.* Jika input memiliki *shape*/dimensi (100,300), maka *TimeDistributed* akan mengaplikasikan *fully connected layer* pada tiap langkah waktu yang berjumlah 100. Output dari *dense layer* akan menjadi input lapisan tekahir *CRF layer*. *CRF layer* berfungsi untuk memprediksi label input dengan mempertimbangkan label pada input yang berdekatan. Semisal, label B-S lebih mungkin diikuti dengan label I-S ketimbang label I-P.

Setelah pendefinisian *layer*, *optimizer* untuk model harus didefinisikan. Dalam penelitian ini, *optimizer* ADAM digunakan. *Hyperparameter* dari *optimizer* adalah *learning rate*. Dalam penelitian ini *learning rate* diberi nilai 0.01. Angka *learning rate* yang terlalu besar akan membuat model lebih cepat mencapai *local minima* namun ada kemungkinan model melewati *local minima* karena angka *learning rate* besar. Angka *learning rate* yang kecil sebaliknya akan mencapai *local minima* dalam waktu yang lebih lama tetapi



***Gambar 4. 9** Graph Model BiLSTM*

kemungkinannya kecil model akan melewatkan *local minima*. Setelah mengatur *optimizer,* model harus dikompilasi dengan memasukkan parameter *loss function, optimizer*, dan metrik yang digunakan.

4.3.2.5 Training Model

Model yang sudah dibuat akan diberi input yang telah melalui tahap *preprocessing*. Model akan mencoba mempelajari pola yang ada pada data latih dan melakukan validasi pada setiap akhir *epoch*. Dalam proses *training*, jumlah *epoch* dan *batch size* harus didefinisikan. Untuk mendapat hasil maksimal,

berbagai macam *epoch* dan *batch size* harus dicoba. *Batch size* adalah jumlah sampel dari dataset yang dimasukkan ke dalam model setiap iterasi. *Epoch* adalah satuan yang digunakan ketika seluruh dataset sudah melewati model. Jika terdapat 1000 data dan peneliti mencoba batch size dengan ukuran 500, maka 1 *epoch* akan selesai dalam 2 iterasi. Batch size yang besar memerlukan memori yang lebih besar dan waktu pelatihan menjadi lebih cepat. *Batch size* yang besar juga membuat *update* parameter bobot lebih sedikit terjadi. Dalam penelitian ini, *batch size* akan dicoba dengan nilai 32 dan akan bertambah untuk mencari model terbaik.

Pelatihan dimulai dengan model menerima input berupa *list* x_train, x_val, y_train, dan y_val. Model akan mempelajari data *training*. Pada setiap akhir *epoch*, model akan melakukan validasi dengan data validasi. *Error* yang didapat dari *loss function* akan dipropagasi ke belakang (backpropagation) agar model dapat melakukan *update* parameter bobot. Pada akhir proses *training*, model akan disimpan.

4.3.2.6 Penyimpanan Model

Model akan disimpan di direktori yang ditentukan peneliti. Model akan disimpan di file yang berekstensi .h5. Model .h5 akan digunakan untuk melakukan prediksi dan pengujian.

4.3.2.7 Load Model

Dalam tahap ini, model terbaik yang sudah tersimpan akan dimuat untuk keperluan prediksi dan pengujian. Dalam API *Keras*, sudah terdapat fungsi untuk memuat model sekaligus dengan bobotnya. Setelah proses *load model*, deklarasi fungsi prediksi harus dijalankan untuk menghindari *error*. Setelah fungsi prediksi dideklarasi, model siap digunakan untuk prediksi kalimat.

4.3.2.8 Prediksi

Sebelum diprediksi, kalimat harus melewati *preprocessing* agar kalimat memiliki bentuk input yang sama dengan yang diharapkan model. Model akan menerima input dan melakukan prediksi. Output dari prediksi adalah skor label yang sesuai untuk tiap token.

4.3.3 Pengkodean Model

Pembuatan model dilakukan dengan sistem operasi Ubuntu berbasis Linux. Bahasa pemrograman yang digunakan adalah *python*. Proses pengkodean menggunakan IDE Visual Studio Code. *Library* utama yang digunakan adalah *Keras* yang menggunakan *backend Tensorflow*.

Dalam pembuatan model, target yang harus dicapai peneliti adalah kemampuan generalisasi model yang baik. Model mungkin memiliki kemampuan prediksi yang baik pada data *training* tetapi justru memiliki kemampuan prediksi yang buruk pada data yang belum pernah dilihat model. Untuk menghindari kemampuan prediksi yang buruk, parameter pada tiap *layer* seperti jumlah unit harus didefinisi dengan benar. Fungsi aktivasi pada *layer dense* juga mempengaruhi kemampuan model. *Dropout* dan teknik regularisasi mungkin juga perlu digunakan dalam model untuk mencegah *overfitting*.

4.3.3.1 Kode Pembuatan Model

Tahap awal pembuatan model adalah inisialisasi *embedding* atau *word vector* karena lapisan kedua adalah *layer embedding*. Kode inisialisasi embedding ada pada file main.py sebelum kode pembuatan model. Berikut adalah kode inisialisasi *embedding*:

```
embeddings_index = {}
with open(os.path.join(GLOVE_DIR, 'wiki.id.vec')) as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
    if not UNKNOWN_TOKEN in embeddings_index:
        embeddings_index[UNKNOWN_TOKEN] = np.asarray([random.gauss(0, 0.01) for _ in range(EMBEDDING_DIM)])
print('Found %s word vectors.' % len(embeddings_index))

print('Preparing embedding matrix.')
word_index= word_dict.idx2str
num_words = min(MAX_NUM_WORDS, len(word_index) + 1)
embedding_matrix = np.zeros((num_words, EMBEDDING_DIM))
for i,word  in enumerate(word_dict.idx2str):
    if i >= MAX_NUM_WORDS:
        continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector

# load pre-trained word embeddings into an Embedding layer
# note that we set trainable = False so as to keep the embeddings fixed
embedding_layer = Embedding(num_words,
                            EMBEDDING_DIM,
                            weights=[embedding_matrix],
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=False,mask_zero=True)
```

Gambar 4. 10 *Kode Inisialisasi Embedding.*

Kode pada gambar 4.7 diawali dengan inisialisasi objek. Objek kemudian diisi dengan *pre-trained embedding. Key* pada objek merupakan kata atau token dan *value*-nya adalah nilai *embedding* dari token tersebut. Kemudian objek `embedding_index` digunakan untuk mencari kata atau token yang ada pada *word dictionary*. Bentuk final *embedding* berupa *list* `embedding_matrix`. *Layer embedding* yang akan dijadikan lapisan kedua model diinisialisasi dengan memberi parameter `num_words` yang merupakan ukuran baris *embedding*, `embedding_dim` yang merupakan dimensi *embedding*, `input_length` yang merupakan panjang input, weight yang merupakan bobot untuk *embedding layer*. Parameter `trainable` diset `false` agar bobot *embedding layer* tidak berubah dan `mask_zero` diset `true` agar token yang memiliki representasi integer 0 diabaikan.

Setelah inisialisasi *embedding*, pembuatan model dimulai. Berikut adalah kode untuk pembuatan model:

```
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32',name='sequence_input')

embedded_sequences = embedding_layer(sequence_input)

x = Bidirectional(LSTM(75,return_sequences=True,unit_forget_bias=True,recurrent_dropout=config.recurrent_dropout_prob,dropout=0.5))(embedded_sequences)
x = Bidirectional(LSTM(75,return_sequences=True,unit_forget_bias=True,recurrent_dropout=config.recurrent_dropout_prob,dropout=0.5))(x)
x = Bidirectional(LSTM(75,return_sequences=True,unit_forget_bias= True,recurrent_dropout=config.recurrent_dropout_prob,dropout=0.5))(x)
x = Bidirectional(LSTM(75,return_sequences=True,unit_forget_bias= True,recurrent_dropout=config.recurrent_dropout_prob,dropout=0.5))(x)
td = TimeDistributed(Dense(100,activation='relu'))(x)

crf = CRF(11)
preds = crf(td)

model = Model(inputs=[sequence_input], outputs=[preds])
#model.summary()

adam = optimizers.adam(lr=0.01,clipnorm=3.0)
model.compile(loss=crf.loss_function,
          optimizer=adam,metrics=[crf.accuracy]
          )
print('Training model..')
```

***Gambar 4. 11** Kode Pembuatan Model.*

Pada gambar 4.9, kode diawali dengan lapisan `sequence_input`. Lapisan `sequence_input` adalah lapisan input atau lapisan paling awal yang memiliki parameter `shape`, `dtype`, dan `name`. Parameter `shape` diisi dengan panjang maksimal kalimat yaitu 100 token. Parameter `dtype` diisi dengan int32 karena lapisan ini menerima input integer 32-bit. Parameter `name` diisi dengan nama untuk lapisan ini karena untuk merujuk pada lapisan ini harus dengan nama yang sudah didefinisikan. Selanjutnya, *layer embedding* didefinisikan dengan parameter input dari lapisan sebelumnya yaitu *layer* `sequence_input`. Lapisan selanjutnya adalah layer Bidirectional LSTM. Layer ini menerima input LSTM yang merupakan objek model LSTM. Parameter dari model lapisan LSTM adalah jumlah unit, `return_sequences`, `recurrent_dropout`, dan `dropout`. Parameter `return_sequences` berfungsi untuk menentukan apakah lapisan menghasilkan satu output atau satu sekuen output. Parameter `recurrent_dropout` adalah *dropout* yang diaplikasikan pada *recurrent state* di masing – masing unit. Parameter `dropout` berfungsi untuk mengaplikasikan *dropout* pada input layer ini.

Lapisan TimeDistributed dengan unit 100 dan fungsi aktivasi ReLU akan diaplikasikan pada setiap langkah waktu. Lalu, lapisan yang terakhir adalah lapisan CRF yang memiliki 11 unit *cell* dan akan menghasilkan 11 output. Pembuatan model ini menggunakan fitur *functional API* dari *Keras.* Sebelum menggabungkan semua *layer* menjadi satu, *optimizer* diinisialisasi dengan *learning rate* sebesar 0.01 dan *clipnorm* dengan nilai 3,0. `Clipnorm` berfungsi untuk melakukan *gradient clipping* untuk menghindari permasalahan *exploding gradient* atau *vanishing gradient.* Lapisan digabungkan dengan fungsi `compile()` yang memiliki parameter `loss`, `optimizer` dan `metrics`. Fungsi *loss* yang digunakan adalah *negative log likelihood* dan metrik yang digunakan adalah akurasi Viterbi dan *F1 score.* Parameter `optimizer` diisi dengan *optimizer ADAM* yang sudah diinisialisasi sebelumnya.

4.3.3.2 Kode Training Model

Untuk memulai proses *training*, fungsi `fit()` dari objek model digunakan. Fungsi tersebut memiliki parameter data *training*, label *training*, *batch size*, *epochs*, *validation_data*, dan *callbacks.* Parameter data dan label *training* diberi input *list* `x_train` dan `y_train`.

```
model.fit(x_train,y_train,batch_size=32,
          epochs=config.max_epochs,
          validation_data=(x_val,y_val),
          callbacks=[ZeroPaddedF1Score,ModelCheck])
```

Gambar 4. 12 *Kode untuk Memulai Training.*

Ukuran `batch_size` bisa dimulai dengan nilai kecil. Jika hasil model dengan `batch_size` kecil kurang baik, nilai `batch_size` yang lebih besar bisa digunakan. Parameter `epoch` juga diisi dengan nilai yang eksperimental. Nilai `epoch` bisa dimulai dari angka yang kecil seperti 10 lalu mulai bertambah sampai *loss function* sudah tidak mengalami penurunan yang signifikan. Parameter `validation_data` diisi dengan *list* `x_val` dan `y_val` yang merupakan data validasi dan label validasi. Parameter `callbacks` adalah fitur yang dimiliki *Keras* untuk mengaplikasikan fungsi pada tahap training tertentu. Dalam pelatihan model ini, `callbacks` diberi fungsi `ZeroPaddedF1Score` yang merupakan fungsi untuk mengukur *F1 score* dari data validasi. `ModelCheck` merupakan objek untuk melakukan *checkpoint* pada kondisi tertentu. Dalam proses pelatihan, `ModelCheck` digunakan untuk melakukan *checkpoint* model terbaik dengan memperhatikan akurasi Viterbi agar dapat melanjutkan proses *training* jika model tanpa sengaja terhenti.

4.3.3.3 Kode Prediksi

Untuk melakukan prediksi, fungsi `predict()` dari objek model digunakan. Parameternya adalah teks kalimat yang sudah melalui proses *preprocessing*.

```
preds = model.predict(prepared_text)
preds = np.argmax(preds,axis=-1)
```

Gambar 4. 13 *Kode Prediksi.*

Variabel `preds` digunakan untuk menampung *list* hasil prediksi model. Output dari prediksi adalah *list* dengan dimensi (100,11). Fungsi `np.argmax()` adalah fungsi dari library *numpy* yang mengembalikan nilai tertinggi dari hasil prediksi 11 label tiap token. Parameter dari fungsi `np.argmax()` adalah *list* hasil prediksi dan *axis* yang dicari nilai tertingginya.

4.3.4 Pengujian Model

Pengujian diamati dengan melakukan pengamatan ketika model dalam proses *training* dan ketika model memprediksi data tes. Variabel yang diamati adalah *precision*, *recall* dan skor *F1* model. Berikut ini adalah 2 jenis pengujian yang dilakukan terhadap model:

1. Pengujian *training*

   Pengujian dilakukan dengan mengamati variabel pengamatan model selama proses *training* data dan pengujian data validasi.

2. Pengujian *testing*

   Pengujian dilakukan dengan mengamati akurasi model ketika memprediksi data tes. Data tes merupakan data yang terpisah dari data *training* maupun data validasi. Performa model terhadap data tes dapat memberi gambaran kemampuan model dalam memprediksi data yang belum pernah dipelajari model dalam proses *training*.