Implementasi Metode Convolutional Neural Network Untuk Klasifikasi Teks

Rifqi Fauzi Rahmadzani
Departemen Teknik Elektro dan Teknologi Informasi
Universitas Gadjah Mada
Yogyakarta, Indonesia
email: rifqifauzi@mail.ugm.ac.id

***Abstrak*— Berkembang pesatnya informasi serta besarnya jumlah data teks dalam bentuk digital telah menjadi subjek penelitian yang penting dalam mengekstrak informasi dari data teks tersebut. Hal ini diperlukan suatu sistem otomatisasi yang dapat mengelola dan mengelompokkan data teks tersebut. Pengelompokan data teks dibutuhkan untuk mempermudah dan mempercepat pencarian suatu informasi. Dalam artikel ini bertujuan untuk mengklasifikasikan teks menggunakan *convolutional neural network* (CNN) berdasarkan polaritas positif dan negatif yang meliputi analisis sentimen dan klasifikasi pertanyaan. Dengan melakukan sedikit modifikasi arsitektur untuk dapat digunakan pada *static vector*. Berdasarkan hasil pengujian menunjukkan bahwa metode CNN dengan melakkukan sedikit *tunning hyperparameter* dan *static vector* dapat meraih hasil yang sangat baik pada proses pengujian.**

Kata kunci*—*neural network, convolutional neural network, klasifikasi teks

I. PENDAHULUAN

Teknologi informasi dan komunikasi saat ini berkembang sangat pesat dan telah menjadi bagian penting kehidupan manusia. Hal ini menyebabkan berbagai macam informasi digital terutama dalam bentuk teks setiap hari terus mengalami pertumbuhan dan jumlahnya semakin besar. Mengelola data teks dengan volume yang sangat besar tidak mudah dilakukan karena membutuhkan waktu yang sangat lama, sehingga dibutuhkan suatu sistem otomatisasi yang dapat mengelola dan mengelompokkan data teks tersebut.

Klasifikasi adalah proses untuk menemukan model atau fungsi yang menjelaskan maupun membedakan konsep atau kelas data. Klasifikasi biasanya dibagi menjadi dua fase yaitu fase *learning* dan fase *test*, pada fase *learning*, sebagian data yang telah diketahui kelas datanya diumpankan untuk membentuk model perkiraan, kemudian pada fase *test* model yang sudah terbentuk diuji dengan sebagian data lainnya untuk mengetahui akurasi dari model tersebut. Akurasi yang memenuhi model ini dapat dipakai untuk prediksi kelas data yang belum diketahui [1].

Deep learning adalah tentang belajar beberapa tingkat representasi dan abstraksi yang membantu untuk memahami data seperti gambar, suara, dan teks [2]. *Convolutional neural network* (CNN) merupakan salah satu metode *deep learning* yang dapat diterapkan untuk melakukan klasifikasi dokumen teks. Seiring dengan berkembangnya komputasi menggunakan *Graphical Processing Unit* (GPU) membuat proses pelatihan model pada algoritma CNN juga menjadi lebih cepat [3].

Dalam artikel ini mecoba menerapkan metode *convolutional neural network* untuk klasifikasi teks berdasarkan polaritas positif dan negatif yang meliputi analisis sentimen dan klasifikasi pertanyaan.

II. ANALISA DAN KEBUTUHAN

A. Analisa Kebutuhan

Menganalisa kebutuhan apa saja yang diperlukan dalam mendefinisikan kebutuhan dari sistem yang dibangun diantaranya.

1. Alat

Alat dan bahan yang digunakan untuk klasifikasi teks menggunakan metode CNN adalah sebagai berikut.

Perangkat keras	Perangkat lunak
a. Processor Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.71 GHZ	a. Sistem operasi Windows 10 Pro 64-bit
b. Hard Disk 1 TB, SSD 250 GB	b. Python 3.6.8
c. RAM 8 GB	c. Google Colab
d. GPU NVIDIA Geforce 930MX	d. Library Tensorflow
	e. Library Keras
	f. Library Numpy

2. Bahan

Percobaan ini menggunakan *dataset* dari data ulasan film Rotten Tomatoes yang merupakan salah satu situs paling berpengaruh bagi konsumen dalam memutuskan film mana yang akan mereka gunakan untuk mendapatkan penghasilan [4].

III. SPESIFIKASI RANCANGAN SOLUSI

A. *Data* dan *Preprocessing*

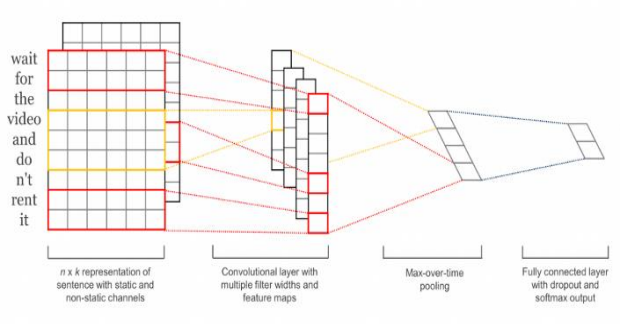
Dataset yang digunakan dari ulasan film Rotten Tomatoes sebanyak 10.662 contoh kalimat ulasan film dan dikelompokkan kedalam dua buah polaritas yaitu positif dan negatif.

Setiap kalimat mempunyai panjang maksimum 59 dengan menambahkan tanda <PAD> untuk semua kalimat agar menjadi 59 kata. Mengeset kalimat dengan panjang yang sama berguna untuk melakukan efisiensi sekumpulan data. Setiap contoh di kumpulan data harus mempunyai panjang yang sama.

Indeks kosakata dibangun dan dipetakan pada setiap kata menjadi bilangan bulat (*integer*) antara 0 dan 18.765 ukuran kosakata. Setiap kata menjadi vektor dari *integer*.

B. Model

Pada percobaan ini menggunakan pola arsitektur model yang ditunjukkan seperti pada Gambar 1.



Gambar 1. Arsitektur model dengan dua saluran untuk contoh kalimat

Layer pertama menyimpan kata-kata dalam sebuah *low-dimensional vector*. Layer selanjutnya menjalankan *convolutions* menggunakan *multiple filter sizes*. Sebagai contoh melakukan pergeseran 3, 4 atau 5 kata dalam satu waktu. Selanjutnya, melakukan max-pool hasil dari layer *convolutional* ke dalam sebuah *long feature vector*, menambahkan *dropout regularization*, dan mengklasifikasikan hasil menggunakan softmax layer [5].

IV. Implementasi

A. Konfigurasi *Hyperparameter*

Untuk dapat memungkinkan berbagai konfigurasi *hyperparameter* maka diperlukan kode pada kelas TextCNN untuk *generating* grafik model difungsi init.

```
1. import tensorflow as tf
2. import numpy as np
3.
4. class TextCNN(object):
5.     """
6.     A CNN for text classification.
7.     Uses an embedding layer, followed by a convolutional, max-pooling and softmax layer.
8.     """
9.     def __init__(
10.       self, sequence_length, num_classes, vocab_size,
11.       embedding_size, filter_sizes, num_filters):

12.         # Implementasi...
```

Gambar 2. Kode program untuk konfigurasi *hyperparameter*

Dalam inisialisasi TextCNN terdapat beberapa argumen sebagaimana berikut.

1. sequence_length merupakan panjang dari kalimat. Dalam hal ini semua kalimat akan diset supaya memiliki panjang yang sama (panjang maksimum 59).
2. num_classes merupakan jumlah dari kelas di output layer yang akan menentukan 2 polaritas yaitu positif dan negatif.
3. vocab_size merupakan ukuran dari kosakata. Hal ini diperlukan untuk mendefinisikan ukuran dari embedding layer, yang akan memiliki bentuk [vocabulary_size, embedding_size].
4. embedding_size merupakan dimensi dari embedding.
5. filter_sizes merupakan jumlah kata yang diinginkan untuk convolutional filters. Sebagaimana num_filters dipilih untuk setiap ukuran spesifik. Sebagai contoh, [3, 4, 5] yang artinya akan mempunyai filter slide 3, 4 dan 5 untuk masing-masing kata, sebagai total dari 3 * num_filters filter.
6. num_filters merupakan jumlah filter untuk setiap ukuran filter.

B. *Input Placeholder*

Dimulai dengan mendefinisikan *input data* yang akan dimasukkan ke *network*.

```
1. # Placeholders for input, output and dropout
2. self.input_x = tf.placeholder(tf.int32, [None, sequence_length], name="input_x")
3. self.input_y = tf.placeholder(tf.float32, [None, num_classes], name="input_y")
4. self.dropout_keep_prob = tf.placeholder(tf.float32, name="dropout_keep_prob")
```

Gambar 3. Kode program untuk *input placeholder*

tf.placeholder membuat variabel *placeholder* yang dimasukkan ke *network* ketika dieksekusi pada saat *train* atau *test time*. Argumen kedua merupakan bentuk dari *input tensor*. *None* berarti bahwa panjang dari dimensi bisa apa saja. Dalam kasus ini, dimensi pertama adalah *batch size*.

Probabilitas berguna untuk mempertahankan *neuron* dalam *layer dropout*, yang juga dilakukan *input* ke *network* karena selama masa pelatihan *dropout* diaktifkan. Setelah itu dinonaktifkan ketika mengevaluasi model.

C. *Embedding Layer*

Layer pertama didefinisikan sebagai embedding layer, yang memetakan indeks kata kosakata dalam representasi vektor dimensi rendah. Hal ini pada dasarnya merupakan tabel yang dipelajari dari data.

```
1. with tf.device('/cpu:0'), tf.name_scope("embedding"):
2.     self.W = tf.Variable(
3.         tf.random_uniform([vocab_size, embedding_size], -1.0, 1.0),
4.         name="W")
5.     self.embedded_chars = tf.nn.embedding_lookup(self.W, self.input_x)
6.     self.embedded_chars_expanded = tf.expand_dims(self.embedded_chars, -1)
```

Gambar 4. Kode program untuk *embedding layer*

Selanjutnya sepasang fitur baru digunakan, yaitu.

1. tf.device("/cpu:0") untuk memaksa operasi dalam mengeksekusinya di CPU. Secara *default* Tensorflow akan mencoba untuk menempatkan operasinya pada GPU jika tersedia, namun untuk implementasi saat ini tidak menggunakan GPU.
2. tf.name_scoper untuk membuat name *scope* baru dengan nama "*embedding*". *Scope* ini akan menambah semua operasi menuju *top-level node* yang disebut "*embedding*" untuk mendapatkan hirarki yang baik ketika memvisualisasikan pada *network* di TensorBoard.

W adalah matrik embedding yang akan dipelajari selama training yang diinisialisasi menggunakan sebuah *random uniform distribution*. tf.nn.embedding_lookup

membuat operasi *actual embedding*. Hasil dari operasi embedding adalah 3 dimensi *tensor of shape* [None, sequence_length, embedding_size].

TensofFlow melakukan operasi *convolutional* conv2d yang mengharapkan *dimentional tensor* dengan dimensi sesuai *batch*, *width*, *height* dan *channel*. Hasil dari *embedding* tidak mengandung *channel dimension*, sehingga akan ditambahkan secara manual, dengan meninggalkan *layer of shape* [None, sequence_length, embedding_size, 1].

D. *Convolution* dan *Max-Pooling Layer*

Pada tahap ini sebuah layer *convolutional* dibangun yang diikuti oleh *max-pooling*. Karena setiap convolutional memproduksi banyak *tensor* dari bentuk yang berbeda yang diperlukan untuk melakukan iterasi pada banyak tensor tersebut. Membuat layer pada setiap tensor, dan kemudian menggabungkan hasilnya ke dalam sebuah fitur vektor yang besar.

```
pooled_outputs = []
for i, filter_size in enumerate(filter_sizes):
    with tf.name_scope("conv-maxpool-%s" % filter_size):
        # Convolution Layer
        filter_shape = [filter_size, embedding_size, 1, num_filters]
        W = tf.Variable(tf.truncated_normal(filter_shape, stddev=0.1), name="W")
        b = tf.Variable(tf.constant(0.1, shape=[num_filters]), name="b")
        conv = tf.nn.conv2d(
            self.embedded_chars_expanded,
            W,
            strides=[1, 1, 1, 1],
            padding="VALID",
            name="conv")
        # Apply nonlinearity
        h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
        # Maxpooling over the outputs
        pooled = tf.nn.max_pool(
            h,
            ksize=[1, sequence_length - filter_size + 1, 1, 1],
            strides=[1, 1, 1, 1],
            padding='VALID',
            name="pool")
        pooled_outputs.append(pooled)

# Combine all the pooled features
num_filters_total = num_filters * len(filter_sizes)
self.h_pool = tf.concat(pooled_outputs, 3)
self.h_pool_flat = tf.reshape(self.h_pool, [-1, num_filters_total])
```

Gambar 5. Kode program untuk *convolution* dan *max-pooling layer*

Di mana, W adalah filter matrik dan h adalah hasil dari pengaplikasian dari *nonlinearity* menuju *convolutional output*. Setiap filter akan melakukan *slides over the whole embedding*, namun berapa kata yang ter-*cover* akan bervariasi. *Padding* "VALID" mempunyai arti bahwa melakukan slide pada filter di atas kalimat tanpa melakukan *padding the edges*, selanjutnya akan melakukan *convolution* yang sifatnya sempit dengan memberikan bentuk output [1, sequence_length - filter_size + 1, 1, 1]. Melakukan *max-pooling* di atas output dari spesifik ukuran filter dan meninggalkan sebuah bentuk *tensor* [batch_size, 1, 1, num_filters]. Hal ini merupakan hal yang esensial sebagai sebuah fitur vektor, dimana dimensi terakhir cocok dengan fitur yang terakhir. Setelah mempunyai semua gabungan bentuk *output tensor* [batch_size, num_filters_total], selanjutnya menggunakan -1 di tf.reshape yang akan melaporkan kepada TensorFlow untuk meratakan dimensi yang ada jika memungkinkan.

E. *Dropout Layer*

Dropout merupakan metode popular untuk membuat *convolutional neural networks* teratur. Sebuah layer *dropout* akan melakukan *stochastic* untuk menonaktifkan pecahan pada neuoran-nya. Hal ini untuk menghindarkan *neuron* dari *co-adapting* dan memaksa neuron tersebut untuk mengaktifkan dengan mendefinisikan berdasarkan *input* dropout_keep_prob untuk *network*. Dalam percobaan ini, akan dilakukan untuk sebuah set dengan 0,5 selama training, dan 1 (menonaktifkan *dropout*) selama evaluasi.

```
# Add dropout
with tf.name_scope("dropout"):
    self.h_drop = tf.nn.dropout(self.h_pool_flat, self.dropout_keep_prob)
```

Gambar 6. Kode program untuk *dropout layer*

F. *Score* dan *Prediction*

Menggunakan fitur vektor dari *max-pooling* (tanpa mengaplikasikan dropout) dapat menghasilkan prediksi berdasarkan *matrix multiplication* dan menghasilkan kelas dengan skor paling tinggi. Selain itu juga dapat mengaplikasikan fungsi *softmax* untuk mengubah skor mentah kepada probabilitas normal, akan tetap tidak akan mengubah prediksi final.

```
# Final (unnormalized) scores and predictions
with tf.name_scope("output"):
    W = tf.get_variable(
        "W",
        shape=[num_filters_total, num_classes],
        initializer=tf.contrib.layers.xavier_initializer())
    b = tf.Variable(tf.constant(0.1, shape=[num_classes]), name="b")
    l2_loss += tf.nn.l2_loss(W)
    l2_loss += tf.nn.l2_loss(b)
    self.scores = tf.nn.xw_plus_b(self.h_drop, W, b, name="scores")
    self.predictions = tf.argmax(self.scores, 1, name="predictions")
```

Gambar 7. Kode program untuk *score* dan *prediction*

Di mana, tf.nn.xw_plus_b merupakan *wrapper* yang bagus untuk menampikan W x + b matrik *multiplication*.

G. *Loss* dan *Accuracy*

Skor yang digunakan dapat mendefinisikan *loss function*. *Loss* merupakan suatu ukuran dari sebuah error yang dibuat oleh *network*, dan tujuannya adalah untuk meminimalisirnya. Standar fungsi *loss* untuk mengkategorisasikan masalah tersebut adalah *cross-entropy loss*.

```
# Calculate mean cross-entropy loss
with tf.name_scope("loss"):
    losses = tf.nn.softmax_cross_entropy_with_logits(logits=self.scores, labels=self.input_y)
```

```
4.      self.loss = tf.reduce_mean(losses) + l2_reg_la
    mbda * l2_loss
```

Gambar 8. Kode program untuk menghitung *loss*

Di mana, tf.nn.softmax_cross_entropy_with_logits merupakan fungsi yang memudahkan untuk melakukan kalkulasi *cross-entropy loss* pada setiap kelas dengan memberikan skor dan input label yang benar. Kemudian akan didapatkan rata-rata dari *losses*. Selain itu, juga dapat mengunakan penjumlahan, namun itu akan membuatnya lebih sulit untuk membandingkan *loss across* dengan ukuran *batch* yang berbeda dan data *train/dev*.

Sebuah ekspresi akurasi juga dapat didefinisikan, yang mana berguna secara kuantitas untuk menjaga *track* selama proses *training* dan *testing*.

```
1.  # Accuracy
2.  with tf.name_scope("accuracy"):
3.      correct_predictions = tf.equal(self.prediction
    s, tf.argmax(self.input_y, 1))
4.      self.accuracy = tf.reduce_mean(tf.cast(correct
    _predictions, "float"), name="accuracy")
```

Gambar 9. Kode program untuk menghitung *accuracy*

H. Prosedur *Training*

Di TensorFlow, sebuah *session* merupakan *environtment* untuk mengeksekusi operasi *graph*, kandungan *state* tentang *variables* dan *queues*. Setiap sesi mengoperasikan pada sebuah *single graph*. Jika secara eksplisit tidak menggunakan session ketika membuat variabel dan operasi, maka akan menggunakan sesi yang sedang berlaku yang dibuat oleh TensorFlow. *Default session* dapat diubah dengan mengeksekusi perintah blok session.as_default() sebagaimana pada kode gambar 10.

Sebuah *graph* berisi operasi dan banyak *tensors*. Multiple graph dapat digunakan pada program, namun kebanyakan program hanya memerlukan *single graph*. *Graph* yang sama dapat digunakan pada *multiple sessions*, tetapi tidak untuk *multiple graphs* pada *one session*. TensorFlow selalu membuat *default graph*, namun dapat juga dibuat sebuah *graph* secara manual dan mengesetnya sebagai default yang baru, seperti yang dilakukan pada kode gambar 10. Secara eksplisit *sessions* dan *graphs* memastikan beberapa sumber dirilis dengan benar ketika tidak dibutuhkan.

```
1.  with tf.Graph().as_default():
2.      session_conf = tf.ConfigProto(
3.        allow_soft_placement=FLAGS.allow_soft_placem
    ent,
4.        log_device_placement=FLAGS.log_device_placem
    ent)
5.      sess = tf.Session(config=session_conf)
6.      with sess.as_default():
7.      # Kode yang beroperasi pada grafik dan sesi de
    fault ada di sini ...
```

Gambar 10. Kode program untuk *training*

Allow_soft_placement mengatur untuk memperbolehkan TensorFlow kembali pada *device* dengan operasi tertentu yang terimplementasi ketika *device* tidak terdeteksi. Sebagai contoh, jika kode ditempatkan operasinya pada sebuah GPU dan ingin menjalankan kode pada sebuah mesin tanpa GPU, jika tidak menggunakan allow_soft_placement akan menghasilkan sebuah *error*. Jika log_device_placement telat diset, TensorFlow akan *log on* sesuai *devices* (CPU atau GPU) tempat operasi dijalankan. Hal ini berguna selama melakukan *debugging*.

I. Pemodelan CNN dan Minimalisasi *Loss*

Ketika ingin memberi contoh model TextCNN semua variabel dan operasi terdefinisikan akan ditempatkan pada *default graph* dan *session* yang telah dibuat.

```
1.  cnn = TextCNN(
2.      sequence_length=x_train.shape[1],
3.      num_classes=y_train.shape[1],
4.      vocab_size=len(vocab_processor.vocabulary_),
5.      embedding_size=FLAGS.embedding_dim,
6.      filter_sizes=list(map(int, FLAGS.filter_sizes.
    split(","))),
7.      num_filters=FLAGS.num_filters,
8.      l2_reg_lambda=FLAGS.l2_reg_lambda)
```

Gambar 11. Kode program untuk pemodelan CNN

Selanjutnya, didefinisikan untuk melakukan optimasi *network loss function*. Tensorflow mempunya beberapa *built-in optimizers*. Sebagaimana pada percobaan ini menggunakan *Adam optimizer*.

```
1.  global_step = tf.Variable(0, name="global_step", t
    rainable=False)
2.  optimizer = tf.train.AdamOptimizer(1e-3)
3.  grads_and_vars = optimizer.compute_gradients(cnn.l
    oss)
4.  train_op = optimizer.apply_gradients(grads_and_var
    s, global_step=global_step)
```

Gambar 12. Kode program untuk minimalisasi *loss*

Di mana, train_op merupakan operasi yang baru dibuat yang mana dapat dijalankan untuk menampilkan *gradient* yang baru dalam parameter. Setiap eksekusi dari train_op adalah tahap *training*. TensorFlow otomatis memperhitungkan variabel mana yang "mudah ditraining" dan dihitung gradien-nya. Dengan mendefinisikan sebuah variabel global_step dan menggunakannya untuk mengoptimalisasi dan memperbolehkan TensofFlow menangani perhitungan dari langkah *training*. Tahapan global akan secara otomatis bertambah setiap train_op dieksekusi.

J. *Summaries*

TensorFlow mempunyai sebuah konsep untuk melakukan ringkasan (*summaries*), yang memperbolehkan untuk menjaga *track* dan memvisualisasikan berbagai kuantitas selama proses *training* dan evaluasi. Sebagai contoh, ketika ingin untuk tetap menjaga *track* agar bagaimana *loss* dan akurasi berkembang dari waktu ke waktu dan juga dapat dengan menjaga track agar lebih kompleks secara kuantitas, seperti histogram dari layer aktivasi. *Summaries* merupakan sebuah serial objek, dan tertulis pada sebuah *disk* menggunakan sebuah *SummaryWriter*.

```
# Output directory for models and summaries
timestamp = str(int(time.time()))
out_dir = os.path.abspath(os.path.join(os.path.curdir, "runs", timestamp))
print("Writing to {}\n".format(out_dir))

# Summaries for loss and accuracy
loss_summary = tf.scalar_summary("loss", cnn.loss)

acc_summary = tf.scalar_summary("accuracy", cnn.accuracy)

# Train Summaries
train_summary_op = tf.merge_summary([loss_summary, acc_summary])
train_summary_dir = os.path.join(out_dir, "summaries", "train")
train_summary_writer = tf.train.SummaryWriter(train_summary_dir, sess.graph_def)

# Dev summaries
dev_summary_op = tf.merge_summary([loss_summary, acc_summary])
dev_summary_dir = os.path.join(out_dir, "summaries", "dev")
dev_summary_writer = tf.train.SummaryWriter(dev_summary_dir, sess.graph_def)
```

Gambar 13. Kode program untuk melakukan *summaries*

Pada kode Gambar 13, secara terpisah melacak *summaries* untuk *training* dan evaluasi. Pada kasus ini merupakan jumlah yang sama, tetapi bisa mempunyai kuantitas yang ingin dilakukan *tracking* hanya selama *training* (seperti parameter mengupdate nilai). tf.merge_summary merupakan sebuah fungsi yang memudahkan dengan menggabungkan operasi *multiple summary* dalam sebuah *single operasi* yang dapat eksekusi.

K. *Checkpointing*

Fitur yang lain dari TensorFlow jika secara khusus ingin menggunakan checkpointing yang berfungsi menyimpan parameter dari model untuk mengembalikannya dilain waktu. *Checkpoints* dapat digunakan untuk melanjutkan *training* pada waktu kemudian, atau ketika memilih parameter *setting* yang terbaik untuk digunakan berhenti lebih awal. *Checkpoints* dibuat menggunakan sebuah objek Saver.

```
# Checkpoint directory
checkpoint_dir = os.path.abspath(os.path.join(out_dir, "checkpoints"))
checkpoint_prefix = os.path.join(checkpoint_dir, "model")
# Tensorflow assumes this directory already exists so we need to create it
if not os.path.exists(checkpoint_dir):
    os.makedirs(checkpoint_dir)
saver = tf.train.Saver(tf.global_variables(), max_to_keep=FLAGS.num_checkpoints)
```

Gambar 14. Kode program untuk melakukan *checkpointing*

L. Inisialisasi Variabel

Sebelum melakukan *train* model, selain itu juga memerlukan inisialisasi variabel pada *graph*.

```
# Initialize all variables
sess.run(tf.global_variables_initializer())
```

Gambar 15. Kode program untuk inisialisasi variabel

Fungsi initialize_all_variables merupakan fungsi yang memudahkan untuk menjalankan semua *initializers* yang telah didefinisikan untuk seluruh variabel.Initializer juga dapat dipanggil melalui variabel secara manual. Hal ini berguna jika ingin melakukan *initialize embeddings* dengan nilai *pre-trained* sebagai contoh.

M. Mendefinisikan Proses *Single Training*

Selanjutnya adalah proses mendefinisikan sebuah fungsi untuk tahapan *single training*, mengevaluasi model pada *batch of data*, dan memperbarui parameter model.

```
def train_step(x_batch, y_batch):
    """
    A single training step
    """
    feed_dict = {
      cnn.input_x: x_batch,
      cnn.input_y: y_batch,
      cnn.dropout_keep_prob: FLAGS.dropout_keep_prob
    }
    _, step, summaries, loss, accuracy = sess.run(
        [train_op, global_step, train_summary_op, cnn.loss, cnn.accuracy],
        feed_dict)
    time_str = datetime.datetime.now().isoformat()
    print("{}: step {}, loss {:g}, acc {:g}".format(time_str, step, loss, accuracy))
    train_summary_writer.add_summary(summaries, step)
```

Gambar 16. Kode program untuk mendefinisikan proses *single training*

feed_dict berisi data untuk *placeholder nodes* yang dilakukan *pass* pada *network*. Nilai untuk seluruh *placeholder nodes* perlu diisi, atau TensorFlow akan memunculkan *error*. Cara lain menangani input data juga dapat menggunakan *queues*.

Selanjutnya yaitu mengeksekusi train_op menggunakan session.run, yang mana mengembalikan nilai untuk semua operasi yang kita minta untuk mengevaluasi. Sebagai catatan, train_op tidak mengembalikan apa-apa, namun hanya memperbarui parameter dari *network*. Pada akhirnya, *loss* dan akurasi ditampilkan dari sejumlah training saat ini dan menyimpan *summaries* pada disk. Perlu diketahui bahwa loss dan akurasi untuk sejumlah training bisa saja secara signifikan berbeda di beberapa jumlah jika *batch size* sifatnya kecil. Karena percobaan ini menggunakan *dropout*, matrik *training* bisa memulai menjadi lebih buruk daripada matrik evaluasi.

Pada percobaan ini juga terdapat sebuah fungsi yang sama untuk mengevaluasi *loss* dan akurasi pada *arbitrary dataset*, seperti *validation set* atau seluruh *training set*. Secara esensial fungsi ini sama dengan baris kode pada Gambar 16, akan tetapi tanpa operasi *training* dan juga menonaktifkan *dropout*.

```
def dev_step(x_batch, y_batch, writer=None):
    """
    Evaluates model on a dev set
    """
    feed_dict = {
      cnn.input_x: x_batch,
      cnn.input_y: y_batch,
      cnn.dropout_keep_prob: 1.0
    }
    step, summaries, loss, accuracy = sess.run(
        [global_step, dev_summary_op, cnn.loss, cnn.accuracy],
        feed_dict)
    time_str = datetime.datetime.now().isoformat()

    print("{}: step {}, loss {:g}, acc {:g}".format(time_str, step, loss, accuracy))
    if writer:
            writer.add_summary(summaries, step)
```

Gambar 17. Kode program untuk mengevaluasi *loss* dan akurasi pada *arbitrary dataset*

N. Perulangan *Training*

Setelah itu, dilakukan proses perulangan *training*. Dengan melakukan iterasi di atas sekumpulan data, memanggil fungsi train_step pada setiap kumpulan, sesekali mengevaluasi dan melakukan checkpoint pada model.

```
# Generate batches
batches = data_helpers.batch_iter(
    list(zip(x_train, y_train)), FLAGS.batch_size, FLAGS.num_epochs)
# Training loop. For each batch...
for batch in batches:
    x_batch, y_batch = zip(*batch)
    train_step(x_batch, y_batch)
    current_step = tf.train.global_step(sess, global_step)
    if current_step % FLAGS.evaluate_every == 0:
        print("\nEvaluation:")
        dev_step(x_dev, y_dev, writer=dev_summary_writer)
        print("")
    if current_step % FLAGS.checkpoint_every == 0:

        path = saver.save(sess, checkpoint_prefix, global_step=current_step)
        print("Saved model checkpoint to {}\n".format(path))
```

Gambar 18. Kode program untuk perulangan *training*

Di mana, batch_iter merupakan fungsi helper yang menulis pada sekumpulan data, dan tf.train.global_step merupakan fungsi yang membantuk nilai kembali dari global_step.

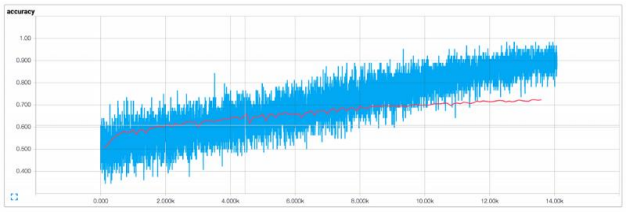
V. Evaluasi

Kode perintah *training* untuk menuliskan *summaries* pada *output directory*, dan menunjuk TensorBoard bahwa direktori tersebut dapat divisualisasikan ke dalam bentuk *graph* dan *summaries* yang telah dibuat.

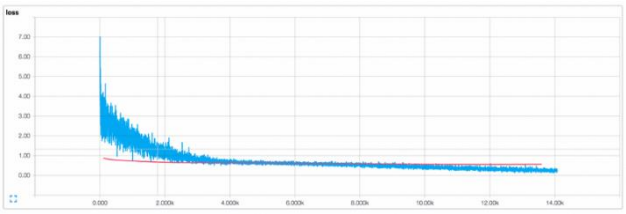
```
tensorboard --
logdir /DIRECTORY_CODE/runs/1561280970/summaries/
```

Gambar 18. Kode perintah untuk menuliskan *summaries* pada *output directory*

Prosedur *training* dijalankan dengan parameter default (128-dimensional embeddings, ukuran filter dari 3, 4 dan 5, dropout dari 0,5 dan 128 filter setiap ukuran filter). Hasil pada akurasi terdapat pada Gambar 19 dan *loss* terdapat pada Gambar 20. Di mana plot biru merupakan training data, sedangkan plot merah merupakan 10% *dev* data.



Gambar 19. Hasil akurasi klasifikasi



Gambar 20. Hasil *loss* klasifikasi

Terdapat beberapa hal pada proses pengujian

Training matrik tidak berjalan mulus karena percobaan ini menggunakan sekumpulan data kecil. Akan tetapi, jika menggunakan sekumpulan data yang lebih besar (atau mengevaluasi pada seluruh *training set*) akan mendapatkan garis biru yang lebih mulus.

Karena akurasi *dev* secara signifikan di bawah akurasi *training* seperti terlihat bahwa *network* mengalami *overfitting* saat *training data*, sehingga perlu lebih banyak data (MR *dataset* sangat kecil), mempunyai regulasi yang lebih kuat, atau mempunyai parameter model yang lebih sedikit. Sebagai contoh, jika akan melakukan eksperimen dengan menambahkan *additional L2 penalties* untuk bobot pada layer terakhir dan memberikan efek *bump up* pada *accuracy* hingga 76%.

Loss training dan akurasi menjadi awal permulaan di bawah metrik *dev* karena menerapkan *dropout*.

VI. Kesimpulan

Dalam artikel ini telah dijelaskan serangkaian percobaan menggunakan *convolutional neural network* dengan melakukan sedikit modifikasi arsitektur untuk dapat digunakan pada *static vector* dan melakukan *tunning* pada *hyperparameter*. Berdasarkan hasil pengujian menunjukkan bahwa metode CNN dapat meraih hasil yang sangat baik pada proses pengujian yang menghasilkan nilai akurasi

mencapai 96.60%. Namun *training* matrik tidak berjalan dengan mulus karena percobaan ini menggunakan data yang kecil. Sehingga percobaan selanjutnya perlu dilakukan seperti menggunakan data yang lebih besar, regulasi yang lebih kuat, mengatur parameter model yang lebih sedikit.

UCAPAN TERIMA KASIH

Penulis ingin mengucapkan terima kasih kepada Tuhan Yang Maha Esa yang telah melimpahkan segala rahmat-Nya sehingga penulis dapat menyelesaikan artikel ini. Penulis juga mengucapkan teima kasih kepada Bapak Noor Akhmad Setiawan yang telah banyak memberikan bimbingan perkuliahan selama ini. Tidak lupa ucapan terima kasih kepada Yoon Kim yang telah memberikan inspirasi atas terselesaikannya penulisan artikel ini.

DAFTAR PUSTAKA

[1] I. Pramudiono, "Pengantar Data Mining : Menambang Permata Pengetahuan di Gunung Data," *Kuliah Umum Ilmu Komputer.com*, pp. 1–4, 2003.

[2] LISA lab, "Deep Learning Tutorials," *Univ. Montr.*, pp. 27–28, 2015.

[3] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A Convolutional Neural Network for Modelling Sentences," *Dep. Comput. Sci. Univ. Oxford*, pp. 655–665, 2014.

[4] B. Pang and L. Lee, "A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts," *Proc. ACL*, 2004.

[5] S. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural Language Processing (Almost) from Scratch," *J. Mach. Learn. Res. 12*, pp. 2493–2537, 2011.