TUGAS AKHIR - KS141501

ANALISIS SENTIMEN TEKS BERBAHASA INDONESIA PADA MEDIA SOSIAL MENGGUNAKAN UNIVERSAL LANGUAGE MODEL FINE-TUNING (ULMFIT) STUDI KASUS: E-COMMERCE

SENTIMENT ANALYSIS FOR INDONESIAN LANGUAGE TEXT ON SOCIAL MEDIA USING UNIVERSAL LANGUAGE MODEL FINE-TUNING (ULMFIT) CASE STUDY: E-COMMERCE

ANDIRA GITA NAWANGSARI
NRP 05211540000051

Dosen Pembimbing
Renny Pradina K., S.T, M.T., SCJP

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

TUGAS AKHIR - KS141501

ANALISIS SENTIMEN TEKS BERBAHASA INDONESIA PADA MEDIA SOSIAL MENGGUNAKAN UNIVERSAL LANGUAGE MODEL FINE-TUNING (ULMFIT) STUDI KASUS: E-COMMERCE

ANDIRA GITA NAWANGSARI
NRP 05211540000051

Dosen Pembimbing
Renny Pradina K., S.T, M.T., SCJP

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

UNDERGRADUATE THESIS - KS141501

SENTIMENT ANALYSIS FOR INDONESIAN LANGUAGE TEXT ON SOCIAL MEDIA USING UNIVERSAL LANGUAGE MODEL FINE-TUNING (ULMFIT) CASE STUDY: E-COMMERCE

ANDIRA GITA NAWANGSARI
NRP 05211540000051

Supervisor
Renny Pradina K., S.T, M.T., SCJP

INFORMATION SYSTEM DEPARTMENT
Information Technology and Communication Faculty
Sepuluh Nopember Institute of Technology
Surabaya 2019

LEMBAR PENGESAHAN

ANALISIS SENTIMEN TEKS BERBAHASA INDONESIA PADA MEDIA SOSIAL MENGGUNAKAN UNIVERSAL LANGUAGE MODEL FINE-TUNING (ULMFIT) STUDI KASUS: E-COMMERCE

TUGAS AKHIR

Disusun Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Departemen Sistem Informasi
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:

ANDIRA GITA NAWANGSARI
NRP. 05211540000051

Surabaya, Juli 2019

KEPALA
DEPARTEMEN SISTEM INFORMASI

Mahendrawati ER, ST, M.Sc, Ph.D
NIP. 197610112006042001

LEMBAR PERSETUJUAN

ANALISIS SENTIMEN TEKS BERBAHASA INDONESIA PADA MEDIA SOSIAL MENGGUNAKAN UNIVERSAL LANGUAGE MODEL FINE-TUNING (ULMFIT) STUDI KASUS: E-COMMERCE)

TUGAS AKHIR

Disusun Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Departemen Sistem Informasi
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Oleh :

ANDIRA GITA NAWANGSARI
NRP. 05211540000051

Disetujui Tim Penguji :Tanggal Ujian :
Periode Wisuda :

Renny Pradina K., S.T, M.T., SCJP **(Pembimbing I)**

Nur Aini Rakhmawati, S.Kom., M.Sc.Eng, Ph.D **(Penguji I)**

Faisal Johan Atletiko, S.Kom., M.Kom. **(Penguji II)**

ANALISIS SENTIMEN TEKS BERBAHASA INDONESIA PADA MEDIA SOSIAL MENGGUNAKAN UNIVERSAL LANGUAGE MODEL FINE-TUNING (ULMFIT) STUDI KASUS: E-COMMERCE)

Nama Mahasiswa : Andira Gita Nawangsari
NRP : 05211540000051
Departemen : Sistem Informasi FTIK-ITS
Pembimbing I : Renny Pradina K., S.T, M.T., SCJP

ABSTRAK

Berdasarkan survey dari Asosiasi Penyelenggara Jasa Internet Indonesia (APJII) pada tahun 2017[1], 54,68% penduduk Indonesia, atau setara dengan 143,26 juta jiwa, telah terpapar oleh internet. Angka ini meningkat dibandingkan tahun sebelumnya. Salah satu layanan yang paling sering diakses oleh pengguna internet adalah media sosial yang diakses oleh sebanyak 87,13% pengguna internet. Memberikan penilaian dan ulasan terhadap produk dan layanan yang telah dipakai pun seringkali dilakukan oleh pengguna media sosial. Selain itu, dalam melakukan pengambilan keputusan, mengetahui pendapat orang lain merupakan salah satu faktor penentu dalam apa keputusan yang akan diambil oleh sebagian besar dari pengguna media sosial. Salah satu objek yang tak luput dalam penilaian dari pengguna media sosial adalah *e-commerce.* Pertumbuhan *online shopper* di Indonesia sendiri selalu naik dari tahun ke tahun, dan diperkirakan mencapai angka 11,9% dari total populasi di Indonesia pada tahun 2018. Adanya fakta ini mendorong untuk dilaksanakannya proses analisis sentimen berdasarkan data yang tersedia pada media sosial, khususnya Twitter. Untuk melakukan analisis sentimen tersebut, pada tugas akhir ini digunakanlah ULMFiT atau Universal Language Model *Fine-tuning* yang merupakan metodologi untuk memanfaatkan *transfer learning* pada pengolahan bahasa alami agar didapatan hasil klasifikasi yang memuaskan hanya dengan dataset yang kecil. Pada penelitian ini, diguanakan tiga jenis skenario untuk pembuatan *pretrained model*, empat jenis pada

proses *fine-tuning*, dan delapan jenis pada pembuatan *classifier* dengan tiga jenis *subtask.* Performa *pretrained model* diperoleh dari model dengan menggunakan data *twitter* sebagai sumbernya, dengan akurasi 53%. Sedangkan *fine-tuning* memiliki performa terbaik pada akurasi 62% menggunakan skenario *full* dan *discriminative learning rates.* Pada klasifikasi pada dua jenis kelas, yaitu positif dan negatif, diperoleh akurasi terbaik 95.76% dengan skenario *full,* pada klasifikasi dengan tiga kelas termasuk kelas netral, diperoleh akurasi 80.86% dengan skenario *full discr,* dan pada klasifikasi dengan lima kelasi diperoleh akurasi 80.56% pada skenario *full.*

Kata Kunci: Analisis, sentimen, twitter, e-commerce, NLP, pretrained model, ULMFiT

SENTIMENT ANALYSIS FOR INDONESIAN LANGUAGE TEXT ON SOCIAL MEDIA USING UNIVERSAL LANGUAGE MODEL FINE-TUNING (ULMFIT) CASE STUDY: E-COMMERCE

Name : Andira Gita Nawangsari
NRP : 05211540000051
Department : Information System FTIK-ITS
Supervisor : Renny Pradina K., S.T, M.T., SCJP

ABSTRACT

Based on a survey of the Indonesian Internet Service Providers Association (APJII) in 2017[1], 54.68% of Indonesia's population, or equivalent to 143.26 million people, has been exposed to the internet. This number increased compared to the previous year. One of the most frequently accessed services by internet users is social media which is accessed by 87.13% of internet users. Providing ratings and reviews of products and services that have been used is often done by users of social media. In addition, in making decisions, knowing the opinions of others is one of the determining factors in what decisions will be made by the majority of social media users. One object that does not escape the assessment of social media users is e-commerce. The growth of online shopper in Indonesia itself always increases from year to year, and is estimated to reach 11.9% of the total population in Indonesia in 2018. This fact encourages the implementation of sentiment analysis process based on available data on social media, especially Twitter. To carry out this sentiment analysis, ULMFiT or Universal Language Fine-tuning Model is used as a methodology to utilize transfer learning in natural language processing so that satisfactory classification results are obtained with only a small dataset. In this study, three types of scenarios were used for the manufacture of pretrained models, four types of fine-tuning processes, and eight types in making classifiers with three types of subtasks. The pretrained model performance was obtained from the model using Twitter data as the source, with 53% accuracy. While fine-tuning has the best performance on 61.73% accuracy using 'full' and 'discriminative learning rates'

scenario. On the classification of two types of classes: positive and negative, obtained the best accuracy 95.76% with 'full' scenario, on the classification of three classes, including a class of neutral, obtained accuracy of 80.86% with 'full discr' scenario, and on the classification with five classes was obtained accuracy of 80.56% with 'full' scenario.
Keywords: Sentiment analysis, twitter, e-commerce, NLP, pretrained model, ULMFiT

KATA PENGANTAR

Dengan mengucapkan rasa syukur kepada Tuhan Yang Maha Pengasih dan Maha Penyayang atas izin-Nya penulis dapat menyelesaikan buku yang sederhana ini dengan judul Analisis Sentimen Teks Berbahasa Indonesia Pada Media Sosial Menggunakan Universal Language Model Fine-Tuning (ULMFiT) Studi Kasus: E-Commerce. Dalam penyelesaian Tugas Akhir ini, penulis diiringi oleh pihak-pihak yang selalu memberi dukungan, saran, dan doa sehingga penelitian berlangsung dengan lancar. Secara khusus penulis mengucapkan terima kasih dari lubuk hati terdalam kepada:

1. Allah SWT, yang dengan rahmat dan hidayahnya membimbing penulis dalam segala aspek kehidupan.
2. Ibu Sri Ekowati dan Bapak Sigit Widharta, serta adik Muhammad Galih Diwangkara yang selalu menjadi semangat penulis dalam menyelesaikan tugas akhir.
3. Ibu Mahendrawati ER, ST, M.Sc, Ph.D selaku Ketua Departemen Sistem Informasi ITS Surabaya.
4. Ibu Renny Pradina K., ST, MT, SCJP selaku dosen pembimbing yang telah mencurahkan segenap tenaga, waktu dan pikiran dalam penelitian ini, serta memberikan motivasi yang membangun.
5. Ibu Nur Aini Rakhmawati S.Kom, M.Sc.Eng, Bapak Radityo Prasetianto W., S.Kom, M.Kom, dan Bapak Faisal Johan Atletiko, S.Kom, M.Kom selaku dosen penguji yang telah memberikan kritik dan saran yang membuat kualitas penelitian ini lebih baik lagi.
6. Ibu Hanim Maria Astuti, S.Kom., M.Sc selaku dosen wali dari penulis yang telah memberikan dukungan dan saran selama penulis menempuh perkuliahan.
7. Magrid, Evia, Yoga, Wenny, Salim Azzam, Supri, dan Faris yang telah menjadi teman seperjuangan penulis dalam mengerjakan tugas akhir.

8. Faro, Ratih, Fefe, Sarah dan Irshad yang telah memberikan motivasi penulis dan menemani penulis dalam menyelesaikan tugas akhir.
9. Rekan-rekan Penghuni IDDA dan Lannister, terimakasih atas dukungan serta pahit manis kehidupan yang telah diajarkan selama ini, terimakasih atas persahabatan yang telah diberikan.
10. Ong Seongwoo dan Kang Daniel, *Wanna One* serta *Wannable*, terimakasih atas perhatian dan dorongannya agar penulis dapat menyelesaikan karya tulis ini dengan cepat.
11. Kelompok *Neo Culture Technology*, terutama Johnny Seo, terimakasih telah menemani dan menghibur penulis dalam menyelesaikan tugas akhir.
12. Mbak Sza, Mbak Mon, dan Mas Bim yang telah mencurahkan perhatian kepada penulis dan membimbing penulis.

Penyusunan tugas akhir ini masih jauh dari kata sempurna, untuk itu penulis menerima segala kritik dan saran yang membangun sebagai upaya menjadi lebih baik lagi ke depannya. Semoga buku tugas akhir ini dapat memberikan manfaat untuk pembaca.

Surabaya, Juli 2019
Penulis

DAFTAR ISI

LEMBAR PENGESAHAN i
ABSTRAK iii
ABSTRACT v
KATA PENGANTAR vii
DAFTAR ISI ix
DAFTAR GAMBAR xiii
DAFTAR TABEL xv
DAFTAR KODE xvi
1 BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 3
1.3 Batasan Permasalahan 3
1.4 Tujuan 4
1.5 Manfaat 4
1.6 Relevansi 4
2 BAB II TINJAUAN PUSTAKA 5
2.1 Penelitian Sebelumnya 5
2.2 Dasar Teori 8
2.2.1. Analisis Sentimen 8
2.2.2. Machine Learning 10
2.2.3. Deep Learning 10
2.2.4. Neural Network 11
2.2.5. Natural Language Processing 11
2.2.6. Transfer Learning 12
2.2.7. Word Embedding 13
2.2.8. ULMFiT 13
2.2.8.1 *Discriminative Fine-Tuning* 15
2.2.8.2 *Slanted Triangular Learning Rates* 16
2.2.8.3 *Gradual Unfreezing* 17
2.2.8.4 *Concat Pooling* 18
2.2.8.5 *BPPT for Text Classification (BPT3C)* 18
2.2.8.6 Bidirectional Language Model 19
2.2.9. Fast.ai 19
2.2.10. LSTM 19
2.2.11. Media Sosial 25

2.2.12. E-Commerce 25
2.2.13 Evaluasi Model 26
3 BAB III METODOLOGI 29
3.1 Diagram Metodologi 29
3.2 Arsitektur Penelitian 29
3.3 Uraian Metodologi 31
3.2.1. Studi Literatur 31
3.2.2. Pengumpulan Data 31
3.2.3. *Preprocessing* Data 32
3.2.4. Penyaringan Data 32
3.2.5. Pelabelan Data 32
3.2.6. Pra-proses Data 34
3.2.7. Proses Pembuatan *Language Model* dan *Classifier* 35
3.2.8. Proses *Pre-train Language Model* 35
3.2.4.1 Proses *Fine-tune Language Model* 35
3.2.4.2 Proses *Fine-tune Classifier* 35
3.2.9. Analisis Model 35
3.2.10. Penyusunan Tugas Akhir 36
4 BAB IV PERANCANGAN 37
4.1 Akuisisi Data Media Sosial 37
4.2 Perancangan Penggalian Data Media Sosial 39
4.3 Perancangan *Pre-processing* Data Media Sosial 39
4.3.1 Perancangan Ekstraksi Tweet 39
4.3.2 Perancangan Filtering Bahasa 40
4.3.3 Perancangan Pembersihan Data 40
4.3.4 Pra-pemrosesan dan Pelabelan Data Pengklasifikasi 41
4.4 Akuisisi Data Wikipedia 43
4.5 Perancangan *Pre-processing* Data Wikipedia 44
4.6 Perancangan Model ULMFiT 44
4.6.1 Perancangan *Pretrained Model* ULMFiT 44
4.6.1.1 Perancangan Dataset *Pretrained Model* 45
4.6.1.2 Perancangan Pretrained Model 45
4.6.2 Perancangan *Finetune* ULMFiT 48
4.6.2.1 Perancangan Dataset *Language Model* 48

4.6.2.2 Perancangan *Fine-tuning Language Model* 48
4.6.3 Perancangan Classifier ULMFiT 49
4.6.3.1 Perancangan Dataset Pengklasifikasi . 49
4.6.3.2 Perancangan Pembuatan Pengklasifikasi 50
4.7 Perancangan Proses Evaluasi Pengklasifikasi 51
5 BAB V IMPLEMENTASI 52
5.1 Lingkungan Implementasi 53
5.2 Pelaksanaan Akuisisi Data Media Sosial 54
5.3 Pelaksanaan *Pre-processing Dataset* Media Sosial 56
5.3.1 Ekstraksi *Tweet* 57
5.3.2 Penyaringan Bahasa 58
5.3.3 Pembersihan Data 59
5.3.4 Pembuatan Databunch 61
5.4 Pelaksanaan Akuisisi Data Wikipedia 66
5.5 Pelaksanaan Pre-processing Data Wikipeda 68
5.5.1 Pembagian Dataset *Wikipedia* 68
5.5.2 *Pre-processing Dataset Wikipedia* 72
5.6 Pembuatan Model ULMFiT 74
5.6.1 Pembuatan *Pretrained Model* 74
5.6.2 Pelaksanaan *Fine-tuning Language Model* 78
5.6.3 Pembuatan *Classifier* 83
5.7 Evaluasi Pengklasifikasi 87
6 BAB VI HASIL DAN PEMBAHASAN 92
6.1 Hasil Perolehan Data Media Sosial 93
6.2 Hasil Perolehan Data Wikipedia 93
6.3 Hasil Pembuatan Pretrained Model 93
6.4 Hasil Pelaksanaan *Fine-tuning Language Model* 97
6.4.1 Konfigurasi Parameter Awal 97
6.4.2 Skenario *Full* 97
6.4.3 Skenario *Full + Discr* 98
6.4.4 Skenario *Full + STLR* 99
6.4.5 Skenario *Full + Discr + STLR* 100
6.4.6 Perbandingan Seluruh Skenario 101
6.5 Hasil Pembuatan *Classifier* 101
6.5.1 Konfigurasi Parameter Awal 101

6.5.2 Subtask A 102
6.5.3 Subtask B 107
6.5.4 Subtask C 112
6.5.5 Pembahasan Hasil Percobaan Tiga Subtask 118
7 BAB VII KESIMPULAN DAN SARAN 125
7.1 Kesimpulan 125
7.2 Saran 126
DAFTAR PUSTAKA 127
LAMPIRAN A: DATA KLASIFIKASI 130
LAMPIRAN B: LOSS DAN AKURASI PETRAINED MODEL 134
LAMPIRAN C: LOSS DAN AKURASI FINETUNING LANGUAGE MODEL 136
LAMPIRAN D: LOSS DAN AKURASI CLASSIFIER SUBTASK A 138
LAMPIRAN E: CONFUSION MATRIX SUBTASK A 147
LAMPIRAN F: LOSS & AKURASI CLASSIFIER SUBTASK B 151
LAMPIRAN G: CONFUSION MATRIX SUBTASK B 160
LAMPIRAN H: LOSS & AKURASI CLASSIFIER SUBTASK C 164
LAMPIRAN I: CONFUSION MATRIX SUBTASK C 173
LAMPIRAN J : F1, AKURASI, PRESISI, DAN RECALL SELURUH SUBTASK 177
BIODATA PENULIS 180

DAFTAR GAMBAR

Gambar 2.1 Tiga Tahapan ULMFiT[8] 14
Gambar 2.2 Slanted triangular learning rate yang digunakan pada ULMFiT[8] 17
Gambar 2.3 Modul berulang pada RNN standar yang memiliki satu lapisan (layer)[28] 20
Gambar 2.4 Modul berulang pada LSTM yang memiliki empat layer yang saling berinteraksi[28] 20
Gambar 2.5 Cell state pada LSTM[28] 21
Gambar 2.6 Gates pada LSTM[28] 21
Gambar 2.7 Langkah dalam LSTM (1) [28] 22
Gambar 2.8 Langkah dalam LSTM (2) [28] 23
Gambar 2.9 Langkah dalam LSTM (3) [28] 23
Gambar 2.10 Langkah dalam LSTM (4) [28] 24
Gambar 2.11 Jumlah Pengunjung Web Bulanan (Juta) E-Commerce Indonesia Kuartal 4 2018 26
Gambar 3.1 Metodologi Pengerjaan Tugas Akhir 29
Gambar 3.2 Arsitektur Penelitian 30
Gambar 3.3 Proses Pemberian Label pada Tweet 34
Gambar 4.1 Proses Pelabelan Data 42
Gambar 4.2 Pelaksanaan *pretraining model* 45
Gambar 4.3 Skenario *Fine-tuning Language Model* 49
Gambar 4.4 Skenario *fine-tuning* tiap *classifier* 51
Gambar 6.1 Grafik *Train loss* dan *Valid loss* Model 1 94
Gambar 6.2 Grafik *Valid loss* dan *Train loss* Model 2 95
Gambar 6.3 Grafik *Valid loss* dan *Train loss Finetuning 1* ... 98
Gambar 6.4 Grafik *Valid loss* dan *Train loss Finetuning 2* ... 99
Gambar 6.5 Grafik *Valid loss* dan *Train loss Finetuning 3* . 100
Gambar 6.6 Grafik *Valid loss* dan *Train loss Finetuning 4* . 100
Gambar 6.7 Grafik *Precision, Recall, Accuracy* dan *F-Measure Subtask A* 102
Gambar 6.8 Grafik *Loss Classifier Full Subtask A* 103
Gambar 6.9 *Confusion Matrix Full Subtask A* 103
Gambar 6.10 Grafik *Loss Classifier Full + discr Subtask A* 104
Gambar 6.11 *Confusion Matrix Full + discr Subtask A* 105
Gambar 6.12 Grafik *Loss Classifier Freez + Discr Subtask A* 105
Gambar 6.13 *Confusion Matrix Last Subtask A* 106

Gambar 6.14 Grafik *Precision, Recall, Accuracy,* dan *F-Measure Subtask B* 108
Gambar 6.15 Grafik *Loss Classifier Full Subtask B* 108
Gambar 6.16 *Confusion Matrix Subtask B* 109
Gambar 6.17 Grafik *Loss Classifier Full + discr* 109
Gambar 6.18 *Confusion Matrix Full + discr Subtask B* 110
Gambar 6.19 Grafik *Loss Classifier Freez + Discr Subtask B* 111
Gambar 6.20 *Confusion Matrix Freez + discr Subtask B*.... 111
Gambar 6.21 Grafik *Precision, Recall, Accuracy,* dan *F-Measure* Subtask C 113
Gambar 6.22 Grafik *Loss Classifier Full + discr Subtask C* 114
Gambar 6.23 *Confusion Matrix Full + discr Subtask C* 114
Gambar 6.24 Grafik *Loss Classifier Freez Subtask C* 115
Gambar 6.25 *Confusion Matrix Freez Subtask C* 116
Gambar 6.26 Grafik *Loss Freez + Discr Subtask C* 116
Gambar 6.27 *Confusion Matrix Freez + Discr Subtask C*... 117
Gambar 6.28 Grafik *Loss Classifier Freez + Stlr Subtask A* 119
Gambar 6.29 Grafik *loss Freez + Discr + STLR* pada subtak B 120
Gambar 6.30 Perbandingan Akurasi Terbaik Tiap Subtask . 124

DAFTAR TABEL

Tabel 2.1 Literatur 1 5
Tabel 2.2 Literatur 2 6
Tabel 2.3 Literatur 3 7
Tabel 2.4 Contoh *tweet* beserta sentimennya 9
Tabel 2.5 *Confusion Matrix* 27
Tabel 3.1 Contoh *tweet* dengan kata kunci 32
Tabel 4.1 Contoh Kata Kunci Streaming 37
Tabel 4.2 Kata Kunci Pada Dataset *Twitter* Bertopik 38
Tabel 4.3 Proses Pembersihan Data 40
Tabel 4.4 Jumlah Data per Label 42
Tabel 4.5 Perlakuan Label Pada Subtask 43
Tabel 4.6 Parameter pada objek *language model learner* 46
Tabel 4.7 Parameter proses *fitting* model 47
Tabel 4.8 Skenario fine-tuning classifier 50
Tabel 5.1 Spesifikasi Komputer 53
Tabel 5.2 *Library* pada Tugas Akhir 53
Tabel 6.1 Tabel Parameter Awal *Pretrained Model* 1 93
Tabel 6.2 Parameter *Pretrained Model* 2 95
Tabel 6.3 Perbandingan Akurasi Model 1 & 2 96
Tabel 6.4 Perbandingan Akurasi dan Loss Model Wiki-100 dan Twitter 97
Tabel 6.5 Parameter Awal Proses *Fine-tuning* 97
Tabel 6.6 *Train loss, Valid loss,* dan Akurasi dari Language Model 101
Tabel 6.7 Parameter Awal Pembuatan Classifier 101
Tabel 6.8 Jumlah Data *Subtask* A 102
Tabel 6.9 Data dan Label pada Subtask B 107
Tabel 6.10 Data dan Label pada Subtask C 112
Tabel 6.11 Perbandingan Akurasi dan *F-measure* Seluruh Subtask 118

Halaman ini sengaja dikosongkan

DAFTAR KODE

Kode 5.1 Pendefinisian Akses API dan nama *file* 54
Kode 5.2 Kelas *MyListener* Pada *Streamer* 55
Kode 5.3 Proses Streaming dan Eksepsi..... 56
Kode 5.4 Pengekstraksi Teks Tweet..... 57
Kode 5.5 Penyaringan Data Berbahasa Indonesia 58
Kode 5.6 Fungsi Replace Emoticon 60
Kode 5.7 Fungsi *Replace char* 61
Kode 5.8 Fungsi *Preprocess* 61
Kode 5.9 Deklarasi Variabel Kelas *PrepareTwitter* 62
Kode 5.10 Fungsi *Split Data* Pada Pembuatan *Databunch*.... 63
Kode 5.11 Fungsi *Prepare Databunch*..... 65
Kode 5.12 Fungsi *Prepare* 65
Kode 5.13 Pendeklarasian Variabel pada *Script Prepare Wiki* 66
Kode 5.14 Pengunduhan Data Wikipedia dan Installasi WikiEtractor..... 67
Kode 5.15 Proses Ekstraksi Data Wikipedia 68
Kode 5.16 Fungsi get_texts pada Create_wikitext..... 69
Kode 5.17 Fungsi Write_wikitext 70
Kode 5.18 Method Main pada Create_Wikitext 71
Kode 5.19 Fungsi Build Vocab pada Postprocess Wikitext... 73
Kode 5.20 Fungsi Limit Vocab pada Postprocess_wiki 73
Kode 5.21 Fungsi Replace_numbers pada Postprocess_wiki 74
Kode 5.22 Fungsi Pretrain Model 75
Kode 5.23 Pendeklarasian Variabel Pretrain LM 75
Kode 5.24 Pengambilan Token Train dan Val..... 75
Kode 5.25 Pembuatan Vocabulary pada Pretrain LM 76
Kode 5.26 Penyimpanan Vocabulary Pretrain LM..... 76
Kode 5.27 Pembuatan TextLMDataBunch Pretrain LM 76
Kode 5.28 Pembuatan Objek Language Model Learner..... 77
Kode 5.29 Proses Training Pretrain LM..... 78
Kode 5.30 Penyimpanan Model Pretrain LM 78
Kode 5.31 Pendeklarasian Variabel Finetuning..... 79
Kode 5.32 Pendeklarasian Network Parameter dan Penggunaan GPU 80
Kode 5.33 Pendeklarasian Variabel Nama dan Lokasi Model Finetuning 80

Kode 5.34 Fungsi Set Deterministik Finetuning..... 81
Kode 5.35 Fungsi Fit Triangular Finetuning 81
Kode 5.36 Fungsi lm_fine_tuning..... 83
Kode 5.37 Fungsi fine_tune 83
Kode 5.38 Pendeklarasian Variabel Classifier..... 85
Kode 5.39 Fungsi Classify Full pada Classifier..... 87
Kode 5.40 Fungsi Evaluation 88
Kode 5.41 Fungsi Plot Confusion Matrix..... 89
Kode 5.42 Fungsi Save Confusion Matrix 90
Kode 5.43 Fungsi Classify 90

BAB I
PENDAHULUAN

Pada bab ini akan dijelaskan tentang latar belakang, rumusan masalah, batasan masalah, tujuan dan relevansi dari penelitian yang dilakukan.

1.1 Latar Belakang

Internet kini bukan lagi suatu hal yang asing bagi masyarakat Indonesia. Berdasarkan survey dari Asosiasi Penyelenggara Jasa Internet Indonesia (APJII) pada tahun 2017[1], 54,68% penduduk Indonesia, atau setara dengan 143,26 juta jiwa, telah terpapar oleh internet. Angka ini meningkat dibandingkan tahun sebelumnya yang hanya 132,7 juta jiwa saja. Pertumbuhan pengguna internet di Indonesia ini terus meningkat dari tahun ke tahun. Penggunaannya pun beragam, seperti untuk *chatting*, *search engine*, dan jual beli barang. Salah satu layanan yang paling sering diakses oleh pengguna internet adalah media sosial, dimana menurut survey dari APJII, diakses oleh sebanyak 87,13% pengguna internet.

Internet dan media sosial kini menjadi salah satu media utama untuk masyarakat dalam mengutarakan opini yang berkaitan dengan kehidupan sehari-harinya. Memberikan penilaian dan ulasan terhadap produk dan layanan yang telah dipakai pun seringkali dilakukan oleh pengguna media sosial. Kegiatan tersebut memiliki pengaruh terhadap pengguna media sosial lain sebagai pembaca, karena dalam melakukan pengambilan keputusan, mengetahui pendapat orang lain merupakan salah satu faktor penentu dalam apa keputusan yang akan diambil oleh sebagian besar dari pengguna media sosial[2]. Dengan adanya fakta ini, sudah sewajarnya bahwa perusahaan maupun pihak-pihak penyedia produk dan layanan tersebut memperhatikan bagaimana ulasan serta penilaian dari penggunanya di media sosial. Karena pada dasarnya, setiap perusahan akan merasa cemas terhadap penerimaan produk atau jasa tersebut[3] sebab memiliki pengaruh terhadap keuntungan yang perusahaan peroleh. Apalagi, konsumen baru memiliki kecenderungan untuk mendengarkan pendapat dari pengguna

lain yang telah terlebih dahulu memakai produk atau layanan tersebut[4]. Selain untuk menilai sentimen untuk produknya sendiri, perusahaan juga dapat mengamati bagaimana kondisi sentimen masayarakat terhadap kompetitor agar dapat mengetahui kekuatan serta kelemahannya. Adanya alasan-alasan tersebut pun mendorong penyedia produk dan layanan untuk melakukan analisis dari sentimen yang diterimanya di media sosial. Salah satu objek yang tak luput dalam penilaian dari pengguna media sosial adalah *e-commerce.* Pertumbuhan *online shopper* di Indonesia sendiri selalu naik dari tahun ke tahun, dan diperkirakan mencapai angka 11,9% dari total populasi di Indonesia pada tahun 2018[5]. Hal ini juga ditandai dengan jumlah pengunjung *online shop* terbanyak pada triwulan IV 2018 yang mencapai angka 168 juta pada *online shop* Tokopedia, meningkat 10% dari kuartal sebelumnya. Diikuti oleh Bukalapak (116 juta) yang meningkat 20 juta pengunjung, dan Shopee (67.7 juta) yang juga meningkat sebesar 29 juta. Tingginya angka pembeli online di Indonesia serta pertumbuhannya yang pesat membuat para penyedia layanan *e-commerce* berlomba-lomba untuk memberikan pelayanan terbaik dibandingkan kompetitornya.

Disisi lain, media sosial *Twitter* merupakan salah satu situs yang paling banyak dikunjungi di dunia[6], dan tak jarang pengguna *Twitter* menguraikan sentimennya mengenai produk dan layanan yang diperoleh, termasuk pengalaman yang diperoleh dari *e-commerce*. Adanya fakta ini mendorong untuk dilaksanakannya proses analisis sentimen berdasarkan data yang tersedia pada media sosial tersebut. Analisis sentimen sendiri merupakan suatu kegiatan yang berkaitan dengan pengolahan komputasional opini, sentimen, dan objektifitas dari suatu teks[2]. Dengan melakukan analisis sentimen, penyedia layanan *e-commerce* dapat melihat secara nyata kecenderungan masyarakat pengguna sosial media *twitter* terhadap produk dan layanannya, serta melihat tren apa yang ada di pasar, sehingga *e-commerce* dapat memberikan inovasi yang menarik bagi pengguna. Selain itu, *e-commerce* juga dapat melihat performa pelayanan yang diberikan selama ini. Pengguna akan memiliki kecenderungan untuk menceritakan pengalamannya, dan ini

dapat digunakan sebagai celah untuk *e-commerce* untuk melihat apakah pengalaman penggunanya membawa citra positif atau justru sebaliknya. Dalam pelaksanaannya, analisis sentimen membutuhkan data yang besar agar pengklasifikasi mampu belajar untuk melakukan klasifikasi sentimen dengan lebih akurat. Padahal, data yang tersedia jumlahnya terbatas. Sehingga, dibutuhkan solusi untuk dapat melakukan analisis sentimen dengan akurasi tinggi meskipun hanya dengan menggunakan jumlah data yang terbatas.

Berdasarkan uraian tersebut, tugas akhir ini memberikan solusi berupa analisis sentimen pengguna media sosial terhadap penggunaan *e-commerce* yang ada di Indonesia, yang dalam tugas akhir ini dibatasi menjadi lima *e-commerce* teratas berdasarkan pada data dari iPrice Group pada kuartal ke empat tahun 2018[7]. Pada tugas akhir ini pula, akan dimanfaatkan ULMFiT atau Universal Language Model Fine-tuning yang merupakan metodologi untuk memanfaatkan *transfer learning* pada pengolahan bahasa alami agar didapatan hasil klasifikasi yang memuaskan hanya dengan dataset yang kecil[8].

1.2 Rumusan Masalah

Berikut merupakan rumusan masalah yang akan diselesaikan dalam tugas akhir ini:

1. Bagaimana cara melakukan analisis sentimen untuk mengetahui sentimen dari pengguna *e-commerce* di media sosial *Twitter* dengan menggunakan *Universal Language Model Fine-tuning* (ULMFiT)*?*
2. Bagaimana performa *language model* Bahasa Indonesia yang dibangun dengan menggunakan *Universal Language Model Fine-tuning* (ULMFiT)?

1.3 Batasan Permasalahan

Berikut merupakan batasan masalah dalam pengerjaan tugas akhir ini:

1. Teks yang akan diolah berupa teks dalam Bahasa Indonesia

2. Studi kasus yang diangkat melingkupi bidang *e-commerce*, dengan objek lima *e-commerce* terbesar di Indonesia menurut data dari iPrice Group
3. Korpus yang digunakan pada tahapan pelatihan berasal dari Wikipedia Bahasa Indonesia serta dari media sosial *twitter*

1.4 Tujuan

Berdasarkan uraian rumusan masalah dan batasan masalah yang telah disebutkan sebelumnya, maka tujuan yang ingin dicapai dalam pembuatan tugas akhir ini adalah mengetahui cara melakukan analisis sentimen untuk mengetahui sentimen dari pengguna *e-commerce* yang ada di Indonesia pada media sosial Twitter dengan memanfaatkan metodologi *Universal Language Model Fine-tuning* (ULMFiT) serta mengetahui kinerja dari model yang dibuat dengan metodologi ULMFiT.

1.5 Manfaat

Berikut manfaat yang diharapkan akan diperoleh dari hasil pengerjaan tugas akhir ini:

Bagi penulis:

- Memahami proses analisis sentimen teks berbahasa Indonesia dengan menggunakan metodologi *Universal Language Model Fine-tuning* (ULMFiT)

Bagi pengetahuan:

- Mengetahui bagaimana performa ULMFiT sebagai metodologi dalam analisis sentimen pada teks berbahasa Indonesia

1.6 Relevansi

Tugas akhir ini berkaitan dengan mata kuliah dari Departemen Sistem Informasi, yaitu Sistem Cerdas, Penggalian Data dan Analitika Bisnis, serta Pengolahan Bahasa Alami yang merupakan mata kuliah bidang keilmuan Laboratorium Akuisisi Data dan Diseminasi Informasi.

BAB II
TINJAUAN PUSTAKA

Pada bab ini berisi penelitian sebelumnya yang digunakan sebagai acuan dalam pengerjaan tugas akhir ini beserta dasar teori yang menunjang penelitian yang ada pada tugas akhir.

2.1 Penelitian Sebelumnya

Pada bagian ini, akan disebutkan penelitian sebelumnya yang digunakan sebagai referensi dalam pengerjaan tugas akhir. Tabel 2.1 – 2.3 telah merangkum kesimpulan dari masing-masing literatur.

Tabel 2.1 Literatur 1

Judul Penelitian	*Universal Language Model Fine-tuning for Text Classification*
Identitas Peneliti	Jeremy Howard, Sebastian Ruder; 2018
Metode	ULMFiT
Kesimpulan	*Paper* ini dibuat untuk mengusulkan suatu metodologi bernama ULMFiT atau *Universal Language Model Fine-tuning for Text Classification* yang memanfaatkan *transfer learning* pada NLP. Metode ini dimulai dengan melakukan *unsupervised pre-training* pada *language model* dengan menggunakan data dari korpus dengan domain umum untuk mempelajari distribusi dari suatu bahasa. Kedua, melakukan *fine-tuning* pada *language model* dengan menggunakan data pada target tugas. Pada tahapan ini akan digunakan *discriminative fine-tuning* dan *slanted triangular larning rates*. Ketiga, melakukan *fine-tuning* pada *classifier* tugas yang ditargetkan dengan menggunakan *gradual*

	unfreezing, *Discr*, dan *STLR*. Metodologi ini kemudian diterapkan pada tiga jenis tugas NLP, yaitu analisis sentimen, klasifikasi pertanyaan, dan klasifikasi topik, dengan hasil *validation error* yang lebih rendah pada hampir seluruh metode yang sebelumnya ada.
Relevansi Penelitian	*Paper* ini akan digunakan sebagai bahan rujukan penulis, dimana penulis akan memanfaatkan metodologi pada ULMFiT untuk melakukan analisis sentimen.
Sumber	[8]

Tabel 2.2 Literatur 2

Judul Penelitian	*ULMFiT at GermEval-2018: A Deep Neural Language Model for the Classification of Hate Speech in German Tweets*
Identitas Peneliti	Kristian Rother, Achim Rettberg; 2018
Metode	ULMFiT
Kesimpulan	*Paper* ini menjelaskan tentang bagaimana ULMFiT dipergunakan pada *Task 1 (Binary Classification)* dari *Germeval Task 2018*. Pada tugas ini, diberikan *tweet* berbahasa Jerman, kemudian diklasifikasikan menjadi mana yang bersifat ofensif dan mana yang tidak. Pada *paper* ini, dipergunakan ULMFiT dan arsitektur AWD-LSTM. *Language model* yang bersifat universal mengalami *pre-train* pada 100 juta artikel dari Wikipedia Bahasa Jerman. Sedangkan *language model* yang lebih spesifik dilatih pada

	303,256 *tweet* yang tidak berlabel. Untuk *classifier*, dilakukan *training* pada *tweet* yang telah diberi label sebelumnya. Dari beberapa model yang dihasilkan, nilai *perplexity* terbaik yang diperoleh adalah 27.39, serta nilai F1 rata-rata adalah 80%.
Relevansi Penelitian	*Paper* ini akan digunakan sebagai referensi penulis dalam membuat model menggunakan ULMFiT serta arsitektur AWD-LSTM
Sumber	[9]

Tabel 2.3 Literatur 3

Judul Penelitian	Regularizing and Optimizing LSTM Language Model
Identitas Peneliti	Stephen Merity, Nitish Shirish Keskar, Richard Socher; 2017
Metode	LSTM
Kesimpulan	Pada *paper* ini, diajukan suatu strategi yang bernama *weight-dropped LSTM*, dimana strategi ini dapat mencegah adanya *overfitting* pada koneksi *recurrent*. *Paper* ini juga melakukan investigasi pada pemakaian rata-rata *SGD (stochastic gradient descent)* dengan pemicu yang tidak monoton untuk *training* dari *language model*, dan hal itu berhasil meningkatkan performa SGD. Selain itu, juga telah diinvestigasi regulasi lainnya, seperti panjang dari BPTT (*backpropagation through time*), *activation regularization* (*AR*), dan *temporal activation regularization* (*TAR*). *Paper* ini selanjutnya menyebut pendekatan yang mereka gunakan sebagai AWD-LSTM

	yang merupakan kependekan dari *ASGD Weight-Dropped LSTM*.
Relevansi Penelitian	Penulis akan menggunakan pendekatan *AWD LSTM* pada *classifier* yang nantinya akan dimanfaatkan untuk melakukan klasifikasi sentimen dari *tweet*, sehingga *paper* ini dijadikan bahan rujukan oleh penulis.
Sumber	[10]

2.2 Dasar Teori

Pada sub bab ini, akan dijelaskan mengenai dasar teori yang akan dimanfaatkan dalam pengerjaan tugas akhir.

2.2.1. Analisis Sentimen

Istilah "analisis sentimen" mulai diperkenalkan melalui jurnal keilmiahan oleh berbagai tokoh pada tahun 2001-2003[3]. Diperkenalkannya istilah ini pada komunitas yang berfokus pada NLP menyebabkan analisis sentimen mendapatkan popularitas. Sejumlah besar makalah yang menyebutkan "analisis sentimen" berfokus pada klasifikasi ulasan menurut polaritasnya, baik positif, maupun negatif. Kini, istilah analisis sentimen atau *sentiment analysis* diartikan sebagai kegiatan untuk mengidentifikasi sentimen dari sebuah teks, yang biasanya meliputi kegiatan menilai apakah suatu teks memiliki sentimen positif, negatif, ataupun netral[11]. Sentimen sendiri dapat bersifat umum maupun spesifik, misalnya mengenai seorang tokoh, sebuah produk, maupun sebuah peristiwa. Salah satu objek yang biasa digunakan sebagai objek untuk melakukan analisis sentimen adalah platform *microblogging* seperti *twitter*, dikarenakan kini banyak pengguna internet yang menulis sentimennya mengenai produk dan layanan yang mereka peroleh, bahkan mengenai pandangan politik dan agama dalam platform tersebut[12]. Pada tabel 2.4, telah dicantumkan tiga *tweet* beserta sentimennya.

Tabel 2.4 Contoh *tweet* beserta sentimennya

Tweet	**Sentimen Keseluruhan**
Pengen cerita pengalaman **puas** dengan **Blibli**. Browsing dan transaksi Jumat jam 10 malem. Pilih kurir yang 3-5 hari biar gratis. Besoknya Sabtu sebelum jam makan siang udah sampe..... Customer happy itu ketika layanan melebihi ekspektasi.	Sangat Positif
Iya sama min..dari semalem aplikasi @**tokopedia lemot** bgt. Udh coba clear cache sama hapus data Mash juga sampai skrg. Kalo Tokped lite atau desktop gk ada kendala. Mohon dibantu follow up sama bagian terkait.	Negatif
HALLO TEMEN TWITTER KU YANG GIVEAWAY HUNTER, GUA MAU COBA GIVEAWAY NIH YA BERHUBUNG GUA LIAT LIAT BANYAK YG NGELUH KAGAK DAPET GIVEAWAY TERUS NAH SAPA TAU LU MENANG DISINI NIH, GA NYA 200K BUAT 2 ORANG PEMENANG. BEBAS MAU PULSA MAU CHECKOUTIN **SHOPEE** DLL POKOK NYA 1 ORG 100K	Netral

Metode yang dapat dipergunakan untuk analisis sentimen pun beragam, mulai dari deep learning dan neural network seperti CNN dan LSTM, SVM, *Maximum Entropy*,

Logistic Regression, *Random Forest*, *Naïve Bayes*, serta *Conditional Random Fields*[11].

2.2.2. Machine Learning

Machine learning atau pembelajaran mesin merupakan program dalam komputer yang mengoptimalisasi performa dari suatu kinerja sesuai kriteria tertentu, menggunakan data contoh ataupun pengalaman di masa lalu[13]. *Machine learning* adalah subbidang dari *artificial intelligence* atau kecerdasan buatan yang populer dan digunakan secara luas oleh industri untuk memecahkan berbagai persoalan[14]. Istilah *artificial intelligence* sendiri bukan merupakan istilah baru. Adanya istilah ini berawal saat Alan Turing mengajukan pertanyaan "Apakah mesin dapat berpikir?" pada tahun 1950[15]. Semenjak Alan Turing datang dengan konsep *Imitation Game*, fokus dari *artificial intelligence* telah menyebar dalam berbagai area. Melihat banyaknya data berukuran besar yang tersedia sekarang, tak heran pendekatan *data-driven* dari *machine learning* kini menjadi populer[14].

2.2.3. Deep Learning

Metodologi dalam *deep-learning* diartikan sebagai suatu metodologi dalam *representation-learning* dengan berbagai level representasi[16]. Berbagai tingkatan representasi tersebut diperoleh dengan membuat modul sederhana yang bersifat *non-linear*, dimana masing-masing modul dapat mentransformasi representasi tersebut dari suatu tingkatan (dimulai dari masukan mentah) ke dalam tingkatan representasi yang lebih tinggi. *Representation-learning* sendiri merupakan suatu kumpulan metodologi yang memperbolehkan suatu mesin untuk mengolah data mentah dan secara otomatis menemukan representasi yang dibutuhkan untuk melakukan deteksi maupun klasifikasi.

Contohnya ada pada suatu gambar yang direpresentasikan dengan suatu *array* yang berisi nilai-nilai dari piksel pada gambar. Lapisan pertama pada *deep learning* biasanya akan merepresentasikan ada tidaknya suatu tepi pada orientasi atau lokasi tertentu di gambar tersebut. Lapisan kedua biasanya akan melakukan deteksi pada motif-motif yang ditemukan pada penyusunan tertentu dari tepi-tepi gambar, sedangkan lapisan ketiga biasanya mengumpulkan motif-motif yang ada ke dalam kombinasi yang lebih besar dimana dapat merepresentasikan bagian dari objek yang dikenal, sedangkan lapisan selanjutnya akan mendeteksi objek dari kombinasi bagian-bagian tersebut. Aspek kunci dari *deep learning* adalah lapisan-lapisan (*layers*) ini tidak didesain oleh manusia, melainkan dipelajari oleh mesin dari data menggunakan suatu prosedur pembelajaran dengan tujuan yang bersifat umum.

2.2.4. Neural Network

Sebuah *neural network* dalam konteks jaringan syaraf tiruan (*artificial neural network*) pada komputer digital, diartikan sebagai sebuah prosesor terdistribusi pararel dalam ukuran besar yang terdiri dari unit pemrosesan sederhana yang memiliki kecenderungan alami untuk menyimpan pengetahuan *experiential* dan membuatnya dapat digunakan[17]. *Neural network* menyerupai otak manusia dalam dua hal, yaitu pengetahuan yang diperoleh oleh jaringan dari lingkungannya didapatkan melalui proses belajar, dan kekuatan hubungan *interneuron* yang disebut dengan bobot sinaptik, digunakan untuk menyimpan pengetahuan yang telah didapatkan.

2.2.5. Natural Language Processing

Natural language processing (NLP) atau pengolahan bahasa alami adalah suatu pendekatan terkomputerisasi untuk menganalisis teks yang berbasis pada sekumpulan teori dan sekumpulan teknologi[18]. Elizabeth D. Lizzy mendefiniskan

NLP sebagai serangkaian teknik komputasi yang secara teoritis termotivasi untuk menganalisa dan merepresentasikan teks yang muncul secara alami dalam satu atau lebih level dalam analisis linguistic untuk tujuan memperoleh pemrosesan bahasa yang menyerupai manusia untuk berbagai tugas atau penggunaan[18]. Istilah NLP sendiri awalnya disebut sebagai NLU (Natural Language Understanding) pada masa-masa awal AI. Meskipun istilah tersebut masih bisa diterima sebagai tujuan dari NLP, namun tujuan dari NLU belum dapat dicapai. Sistem dari NLU sendiri akan mampu untuk:

- Memparafrase suatu masukan
- Mengubah bahasa dari suatu teks ke bahasa lain
- Menjawab pertanyaan mengenai suatu konten dari teks
- Menarik kesimpulan dari suatu teks

Meskipun NLP telah mampu memberikan terobosan dalam pencapaian tujuan nomor 1 dan 3, namun kenyataan bahwa NLP belum mampu menarik kesimpulan dari suatu teks membuat NLU masih menjadi tujuan utama dari NLP. Selain itu, masih terdapat tujuan praktis yang ingin dicapai oleh NLP, yang banyak berkaitan dengan aplikasi tertentu dimana NLP digunakan.

2.2.6. Transfer Learning

Proses pelatihan dari *neural network* biasanya dimulai dari nol, dan hanya bergantung pada *training data* sebagai acuannya[19]. Namun, karena semakin banyaknya jaringan yang dilatih pada tugas-tugas yang bermacam-macam, maka dicarilah suatu metode untuk menghindari adanya perulangan untuk membangun jaringan dari awal. Suatu contoh adalah pada jaringan *speech recognition* yang hanya dilatih dengan pembicara *American English*. Bagaimanapun pada aplikasi baru, pembicara memiliki aksen *British.* Karena kedua tugas tersebut memiliki kesamaan dalam distribusinya, karena pembicara sama-sama berbahasa Inggris, maka kemungkinan ada suatu

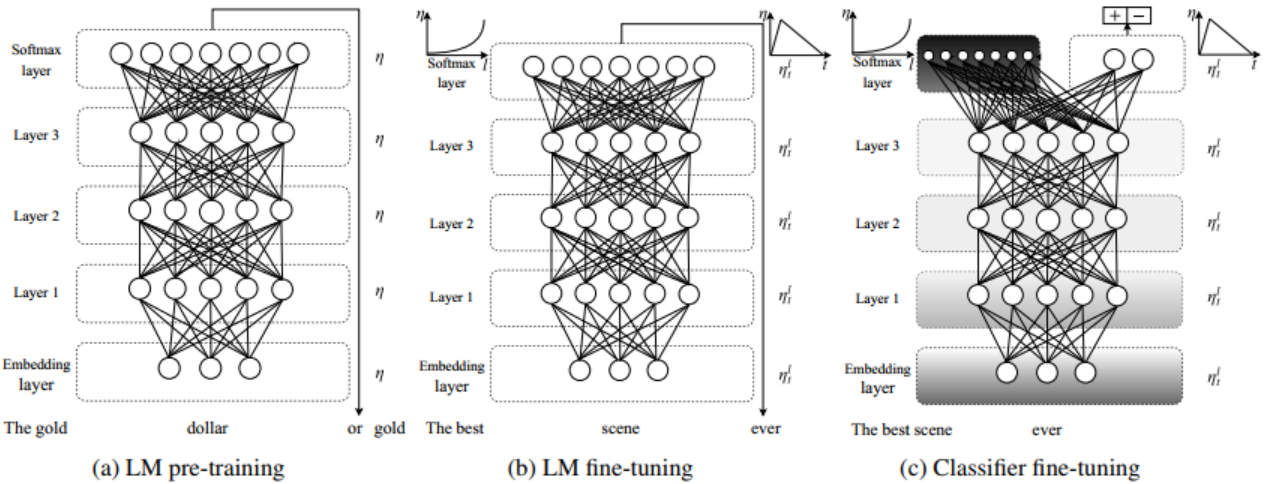
hubungan yang dapat dieksploitasi sehingga proses pembelajaran dapat dipercepat pada jaringan dengan aksen *British*. Hal tersebut merupakan ide awal adanya *transfer leaning*, yang merupakan suatu persoalan dalam penelitian *machine learning* yang berfokus pada penyimpanan pengetahuan atau *knowledge* yang diperoleh sembari memecahkan suatu persoalan dan menggunakannya pada permasalahan yang berbeda namun saling berkaitan[20].

2.2.7. Word Embedding

Word embedding atau yang dikenal juga dengan *dsitibuted word representation*, memiliki kemampuan untuk menyimpan informasi semantik dan sintaktis dari kata-kata yang tersimpan dalam sebuah korpus tak berlabel berukuran besar[21]. Jumlah dari jenis *word embedding* bertambah tiap tahunnya[22]. Suatu model baru biasanya dievaluasi dalam berbagai jenis tugas dalam NLP, dan diakui sebagai suatu perkembangan apabila model tersebut mencapai nilai akurasi yang lebih tinggi. Ada berbagai jenis pengaplikasian yang bisa digunakan untuk tujuan ini, diantaranya *named entity recognition*[23], *semantic role labeling*[24], dan *syntactic parsing*[25]. Meskipun begitu, pengaplikasian pada tugas yang berbeda dapat merujuk pada aspek yang berbeda dari suatu *word embedding*, dalam kata lain performa yang baik pada suatu aplikasi belum tentu menunjukkan performa yang baik pada tugas yang lainnya[22].

2.2.8. ULMFiT

Universal Language Model Fine-tuning (ULMFIT) merupakan suatu metode yang dapat digunakan untuk mendapatkan hasil layaknya *transfer learning* pada *computer vision* yang dapat digunakan pada berbagai jenis tugas untuk NLP[8]. ULMFiT melakukan *pretrains* pada sebuah model bahasa pada korpus dengan domain umum dan melakukan *fine-tunes* dengan target *tast* menggunakan teknik-teknik terbaru.



Gambar 2.1 Tiga Tahapan ULMFiT[8]

ULMFiT memiliki tiga tahapan seperti digambarkan pada gambar 2.1, diantaranya:

1. *Language model* dilatih pada sebuah korpus bahasa dengan domain umum untuk mendapatkan fitur-fitur umum dari sebuah bahasa dalam lapisan-lapisan yang berbeda
2. *Language model* mengalami *fine-tuned* menggunakan data pada target tugas dengan menggunakan *discriminative fine tuning* (‘*Discr*’) dan *slanted triangular learning rates* (SLTR) untuk mempelajari fitur-fitur spesifik dalam suatu tugas
3. Pengklasifikasi *(classifier)* mengalami *fine-tuned* pada target tugas dengan menggunakan *gradual unfreezing*, *‘Discr’,* dan SLTR yang digunakan untuk mempertahankan representasi level rendah dan beradaptasi dengan representasi level tingginya. Pada gambar, langkah yang memiliki baying-bayang disebut *unfreezing stages*, sedangkan yang berwarna hitam disebut *frozen*.

Metode ini dapat bekerja pada berbagai tugas dalam berbagai ukuran dokumen, angka, dan tipe label, hanya memerlukan satu arsitektur dan proses *training*, tidak membutuhkan *preprocessing* khusus, serta tidak membutuhkan tambahan *in-domain* dokumen atau label.

2.2.8.1 *Discriminative Fine-Tuning*

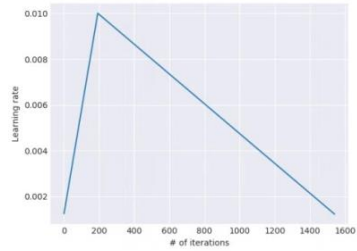
Ide dibalik adanya *discriminative fine-tuning* adalah suatu prinsip dimana lapisan atau *layer* dalam suatu model menyimpan informasi yang berbeda, sehingga dibutuhkan *learning rate* yang berbeda. Sama halnya pada tugas di *language modeling*, lapisan-lapisan atau *layers* pertama akan memiliki informasi paling umum dari suatu bahasa dan akan

membutuhkan *fint-tuning* atau penyesuaian pada bobot yang lebih sedikit. Namun, kebutuhan akan *fine-tuning* ini akan semakin bertambah jika semakin mendekati *layer* terakhir. Sehingga, ULMFiT mengajukan konsep pengunakan *learning rates* yang berbeda pada tiap *layer* yang ada. Hal pertama yang dilakukan adalah dengan melakukan pemilihan terhadap *learning rate* dari *layer* terakhir (η^{L}) dengan hanya melakukan *fine-tuning* pada *layer* tersebut, lalu *learning rates* pada *layer* setelahnya menyesuaikan dengan rumus $\eta^{l-1} = \eta^{l} / 2.6$ dimana *l* adalah *layer* ke *l*.

2.2.8.2 *Slanted Triangular Learning Rates*

Teks target yang akan digunakan dalam klasifikasi memiliki distribusi yang berbeda dengan *language model* yang telah mengalami *training* atau *pre-trained language model*. Tujuan dari *fine-tuning* pada tahapan ini adalah untuk membuat parameter dari model dapat beradaptasi dengan fitur teks pada tugas-tugas yang spesifik. Untuk ini, maka diusulkan *slanted triangular learning rates* (SLTR) dimana *learning rate* awalnya meningkat secara linear dan kemudian menurun secara linear pula. Konsep ini merupakan modifikasi dari konsep *triangular learning rates* dengan penambahan yang pendek dan penurunan yang lebih lama.

Awalan berupa penambahan yang pendek pada *learning rate* dibutuhkan karena kita ingin model untuk secara cepat bertemu dengan wilayah yang cocok pada bidang parameter dari tugas target. Hal ini diikuti dengan penurunan yang lama yang memungkinkan penyempurnaan yang lebih jauh pada parameter-parameter. Karena itu penggunaan *learning rates* yang sama tidak akan cocok.



Gambar 2.2 Slanted triangular learning rate yang digunakan pada ULMFiT[8]

Pembaruan pada SLTR dapat dilihat sebagai berikut:

$$\text{cut} = \lfloor T \cdot cut\_frac \rfloor$$

$$\text{p} = \begin{cases} t/cut, & \text{if } t < cut \\ 1 - \frac{t-cut}{cut \cdot (1/cut\_frac - 1)}, & otherwise \end{cases}$$

$$\eta_t = \eta_{max} \cdot \frac{1 + p \cdot (ratio-1)}{ratio}$$

dimana,

- **T** adalah jumlah iterasi pelatihan
- **cut_frac** adalah fraksi dari iterasi yang akan kita naikkan *learning rates*-nya
- **cut** adalah iterasi dimana kita berpindah dari menambah *learning rates* menjadi mengurangi *learning rates*
- Untuk **t** < **cut**, **p** adalah jumlah iterasi yang mana *learning rates* telah ditambah dari total iterasi penambahan dan, untuk **t >= cut**, **p** adalah total jumlah dari iterasi dimana *learning rate* telah berkurang dari total iterasi pengurangan
- **ratio** menspesifikasi seberapa lebih kecilnya *learning rate* paling kecil dari *learning rate* paling besar, $\boldsymbol{\eta_{max}}$
- $\boldsymbol{\eta_t}$ adalah *learning rate* dari iterasi **t**
- secara umum, **cut_frac** = 0.1, **ratio** = 32 and $\boldsymbol{\eta_{max}}$ = 0.01

2.2.8.3 *Gradual Unfreezing*

Fine-tuning pada pengklasifikasi target sangat sensitif dan *fine-tuning* yang agresif dapat menyebabkan pada hilangnya

manfaat dari *language model pre-training*. Oleh karena itu, diusulkanlah *gradual unfreezing* untuk penyetelan pada pengklasifikasi.

- *Layer* atau lapisan terakhir pada LSTM adalah yang pertama tidak dibekukan (*unfrozen*) dan model disetel untuk satu *epoch*, sedangkan *layer* lain pada LSTM dibekukan.
- Lalu *layer* yang sebelumnya dibekukan dibagian bawah *layer* sebelumnnya dicairkan (*unfroze*)
- Proses ini terus diulangi hingga semua *layer* telah disetel (*fine-tuned)* dan konvergen

2.2.8.4 *Concat Pooling*

Sinyal dalam suatu tugas klasifikasi teks seringkali terkandung dalam beberapa kata, yang bisa saja terjadi dimana saja di dalam dokumen. Karena dokumen masukan dapat terdiri dari ratusan kata, informasi informasi dapat hilang jika kita hanya memperhitungkan *last hidden state* dari suatu model. Dengan alasan ini, *last hidden state* dari suatu model disatukan dengan menggunakan representasi *max-pooled* dan *mean-pooled* dari *hidden state* sebanyak langkah yang dapat disimpan dalam memori GPU $\mathbf{H} = \{h_1,\ldots,h_T\}$:

$$\mathbf{h}_C = [\mathbf{h}_T ; \text{maxpool}(\mathbf{H}); \text{meanpool}(\mathbf{H})]$$

dimana [] adalah suatu rangkaian.

2.2.8.5 *BPPT for Text Classification (BPT3C)*

Serupa dengan *backpropagation through time* (BPTT) untuk *language modeling*, BPTT untuk klasifikasi teks diperkenalkan. Dokumen akan dibagi menjadi potongan-potongan yang lebih kecil dengan ukuran panjang yang telah ditentukan *b*. Pada awal tiap potongan, model diinisiasi dengan hasil akhir dari potongan sebelumnya. *Gradients* merupakan *back propagated* pada *bacthes* yang memiliki *hidden state* yang berkontribusi untuk prediksi akhir. Dalam prakteknya, digunakan *variable length backpropagation sequences.*

2.2.8.6 Bidirectional Language Model

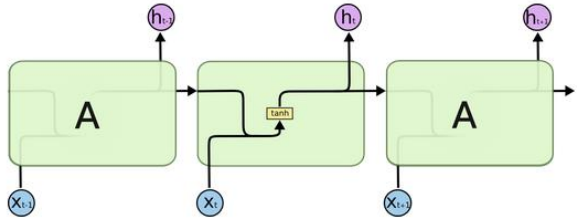
Pada penelitian yang dilakukan untuk perancangan ULMFiT, selain menggunakan *unidirectional language model*, dilakukan juga *pre-train* dengan menggunakan *bidirectional language model*, dimana dengan menggunakan *forward* beserta *backward* dari *language model*.

2.2.9. Fast.ai

Fast.ai adalah suatu laboratorium penelitian yang didirikan oleh Jeremy Howard dan Rachel Thomas[26]. Fast.ai dapat juga dideskripsikan seagai suatu *research lab* yang dilengkapi dengan berbagai materi, serta *library python* yang mudah digunakan dengan komunitas yang luas. *Library* mereka sendiri merangkup berbagai *library* populer mengenai *deep learning* dan *machine learning* serta memiliki tampilan yang mudah untuk digunakan.

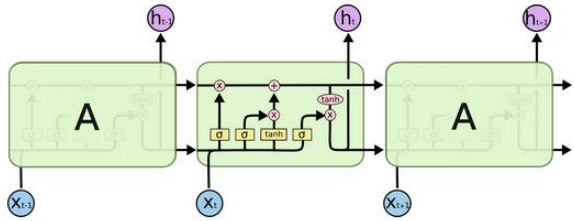
2.2.10. LSTM

Long Short Term Memory atau LSTM pertama kali diperkenalkan oleh Hochreiter dan Schmidhuber pada tahun 1997[27]. LSTM hadir untuk memecahkan persoalan yang ada pada RNN, yaitu ketidakmampuannya untuk mengatasi ketergantungan jangka panjang pada suatu informasi[28]. Semua jenis *recurrent neural networks* (RNN) memiliki bentuk berupa rantai dari modul *neural network* yang berulang. Pada RNN standar, modul yang berulang ini akan memiliki struktur yang sangat sederhana, seperti sebuah *layer* tanh pada gambar 3.



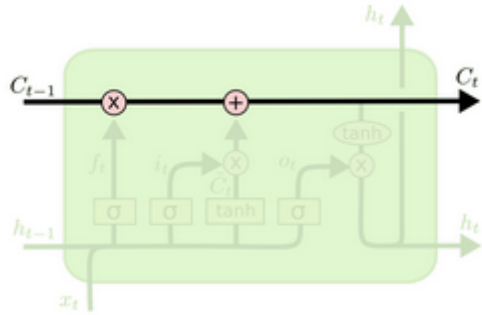
Gambar 2.3 Modul berulang pada RNN standar yang memiliki satu lapisan (layer)[28]

LSTM sendiri juga memiliki struktur yang serupa dengan rantai, namun modul yang berulang di dalamnya memliki struktur yang berbeda. Daripada memiliki satu *layer neural network*, ada empat layer didalamnya yang bekerja dengan cara yang spesial[28], yang digambarkan dengan kotak kuning pada gambar 2.3.



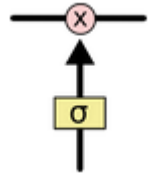
Gambar 2.4 Modul berulang pada LSTM yang memiliki empat layer yang saling berinteraksi[28]

Kunci dari LSTM adalah *cell state*, yang merupakan garis horizontal pada bagian atas diagram, seperti yang telah digambarkan pada gambar 2.4. *Cell state* ini selayaknya *conveyor belt*, yang berjalan lurus di sepanjang rangkaian rantai, dengan hanya sedikit interaksi linear. Sangat mudah bagi informasi untuk berjalan tanpa adanya perubahan.



Gambar 2.5 Cell state pada LSTM[28]

LSTM memiliki kemampuan untuk menghapus dan menambahkan informasi ke dalam *cell state*, yang secara hati-hati diatur oleh suatu struktur yang dinamakan *gates*. *Gates* merupakan suatu jalan untuk secara opsional membiarkan informasi untuk lewat. Sebuah *gates* digambarkan pada gambar 2.5. *Gates* terdiri dari *sigmoid neural net layer* dan sebuah *pointwise multiplication operation*.

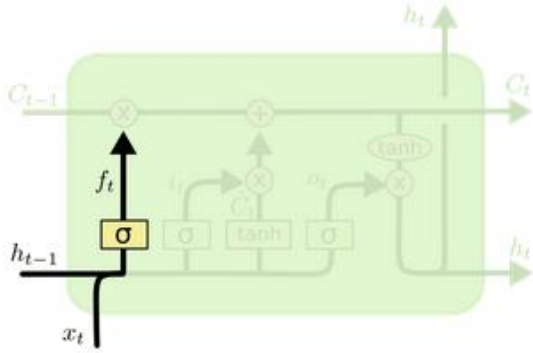


Gambar 2.6 Gates pada LSTM[28]

Sigmoid layer mengeluarkan angka diantara nol dan satu, mendeskripsikan seberapa banyak tiap komponen yang seharusnya dibiarkan lewat. Nilai nol dimaksudkan tidak ada yang boleh lewat, sedang nilai satu dimaksudkan bahwa seluruhnya boleh lewat. Sebuah LSTM memiliki tiga *gates* untuk melindungi dan mengatur *cell state.*

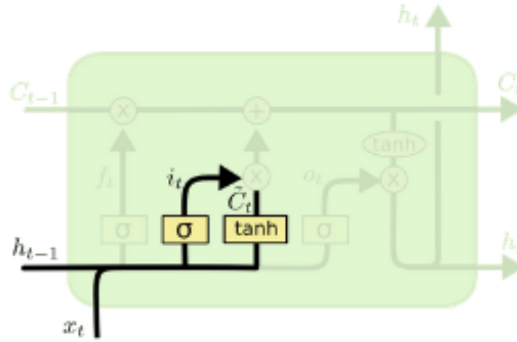
Langkah pertama dalam algoritma LSTM adalah untuk menentukan informasi mana yang akan dibuang dari *cell state*. Keputusan ini diuat oleh sebuah lapisan (*layer)* sigmoid yang disebut *forget gate layer*. Keputusan diambil berdasarkan h_{t-1}

dan x_t seperti pada gambar 2.6, dengan keluaran berupa angka diantara 0 dan 1 di dalam *cell state* C_{t-1}. Angka 1 menandakan bahwa informasi harus disimpan seluruhnya sedangkan 0 menunjukkan bahwa informasi harus dihapus. Proses ini digambarkan pada gambar 2.7, dimana f_t merupakan *forget gate*.



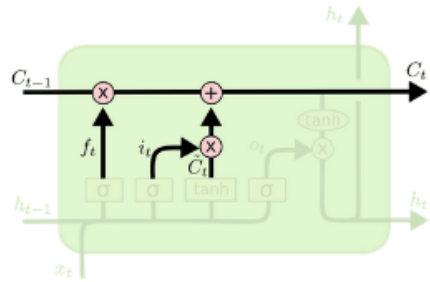
Gambar 2.7 Langkah dalam LSTM (1) [28]

Langkah selanjutnya, seperti yang digambarkan pada gambar 2.8, adalah dengan menentukan informasi baru apa yang akan disimpan di dalam *cell state*. Langkah ini melibatkan dua bagian. Pertama, sebuah *sigmoid layer* yang bernama *input gate layer* menentukan nilai yang mana yang akan diperbarui. Selanjutnya, sebuah *tanh layer* membuat suatu vektor yang menjadi kandidat nilai baru, $\tilde{C}_t$, yang dapat ditambahkan pada *state*. Pada langkah selanjutnya, akan dilakukan pengkombinsian dari dua hal ini untuk pembaruan pada *state.*



Gambar 2.8 Langkah dalam LSTM (2) [28]

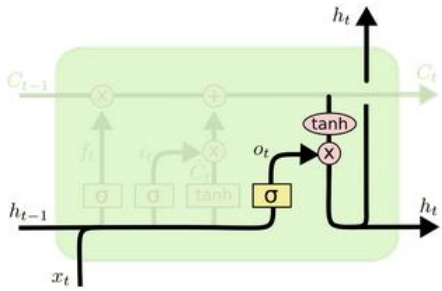
Selanjutnya ada langkah untuk melakukan pembaruan pada *cell state* lama, $C_{t-1,}$ ke dalam sebuah *cell state* baru C_t. Langkah ini telah digambarkan pada gambar 2.9. Langkah selanjutnya telah menentukan langkah apa ang harus dikerjakan, sedangkan langkah ini hanya perlu melaksanakannya secara nyata. Kita akan menggandakan *state* lama dengan f_t, lalu melupakan hal-hal yang telah diputuskan harus dilupakan. Kemudian menambahkan $i_t * \tilde{C}_t$. Ini adalah kandidat nilai baru, diskalakan oleh seberapa banyak keputusan kita untuk melakukan pembaruan nilai dari tiap *state*.



Gambar 2.9 Langkah dalam LSTM (3) [28]

Yang terakhir, kita akan memutuskan apa yang akan menjadi luaran. Luaran ini dapat menjadi dasar dari *cell state*, namun akan menjadi versi yang telah tersaring. Langkah ini

digambarkan pada gambar 2.10. Pertama, kita akan menjalankan suatu *sigmoid layer* yang menentukan bagian-bagian apa dari *cell state* yang akan kita hasilkan. Kemudian, kita akan meletakkan *cell state* melalui *tanh* (untuk mengubah nilai menjadi diantara -1 dan 1) dan mengalikannya dengan luaran dari *sigmoid gate*, sehinga kita hanya melakukan luaran dari bagian yang kita tentukan itu. Luaran disini dapat berupa subjek, kata kerja, dan luaran lainnya yang relevan.



Gambar 2.10 Langkah dalam LSTM (4) [28]

2.2.10.1 AWD-LSTM

AWD-LSTM merupakan singkatan dari ASGD *Weight-Dropped* LSTM, yang merupaan suatu strategi yang menggunakan *DropConnect* dan berbagai strategi lain yang dapat membantu mencegah *overfitting* pada setiap koneksi *recurrent*[10]. Beberapa strategi lain yang diajukan bersamaan dengan AWD-LSTM adalah *weight-dropped* LSTM, *non-monotonically triggered ASGD*, dan beberapa regulasi tambahan yaitu *variable length backpropagation sequences, variational dropout, embedding dropout, weight tying, independent embedding size and hidden size*, dan yang terakhir *Activation Regularization* (AR) *and Temporal Activation Regularization* (TAR). AWD-LSTM sendiri akan digunakan oleh metodologi ULMFiT pada fast.ai sebagai arsitekturnya.

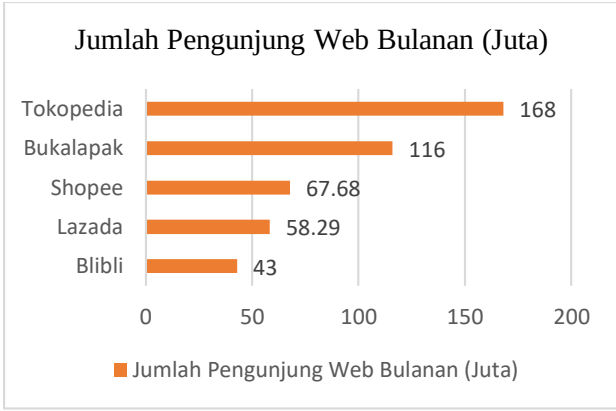
2.2.11. Media Sosial

Social media atau media sosial adalah sebuah kumpulan dari aplikasi berbasis internet yang dibangun diatas fondasi ideologis dan teknis dari Web 2.0, yang mengizinkan pembuatan dan pertukaran dari konten buatan pengguna[29].

Salah satu media sosial yang telah banyak dipergunakan, sekaligus yang akan dimanfaatkan dalam tugas akhir ini adalah Twitter. Twitter merupakan salah satu jejaring media sosial yang memungkinkan pengguna untuk membaca dan menuliskan pesan dengan panjang maksimal 240 karakter per pesannya, yang juga dikenal sebagai kicauan atau *tweet*. Twitter juga menjadi salah satu media sosial yang paling banyak digunakan di Indonesia, dengan pengguna aktif hingga 27% dari jumlah total pengguna internet yang ada[30].

2.2.12. E-Commerce

Electronic commerce, atau juga yang dikenal sebagai *e-commerce* merupakan suatu jenis industry dimana kegiatan jual dan beli dari suatu produk atau layanan menggunakan sistem berupa internet serta jaringan komputer lainnya[31]. Perdagangan elektronik atau *electronic commerce* biasanya melibatkan beberapa jenis teknologi, seperti perdagangan bergerak (*mobile commerce*), pengiriman data elektronik, manajemen rantai pasok, pemasaran internet, pemrosesan transaksi *online*, dan lain sebagainya. Pada tugas akhir ini, *e-commerce* akan digunakan sebagai studi kasus dalam penerapan analisis sentimen, dimana akan digali bagaimana sentimen pengguna media sosial terhadap produk dan layanan dari *e-commerce* terkait, dengan batasan *e-commerce* yang akan diamati adalah 5 *e-commerce* terbesar di Indonesia menurut data dari iPrice Group pada kuartal ke empat tahun 2018, yaitu Tokopedia, Bukalapak, Shopee, Lazada, dan Blibli[7]. Statistik dari pengunjung web bulanan dapat dilihat pada gambar 2.11.



Gambar 2.11 Jumlah Pengunjung Web Bulanan (Juta) E-Commerce Indonesia Kuartal 4 2018

2.2.13 Evaluasi Model

Berikut merupakan beberapa *metrics* yang digunakan sebagai pengevaluasi model dalam tugas akhir ini:

- **Akurasi**

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

- **Presisi**

$$Precision = \frac{TP}{TP + FP}$$

- ***Recall***

$$Recall = \frac{TP}{TP + FN}$$

- **F-*measure***

$$F1 = 2\,x\,\frac{Precision * Recall}{Precision + Recall}$$

dengan *TP* adalah *true positive, FP false positive, TN true negative* serta *FN false negative* seperti yang digambarkan pada tabel 2.5.

Tabel 2.5 *Confusion Matrix*

		Nilai Sebenarnya	
		True	*False*
Nilai Prediksi	*True*	*True Positive*	*False Positive*
	False	*False Negative*	*True Negative*

- ***Loss Function***

  FastAi memanfaatkan *logarithmic loss* untuk menghitung performa dari model pengklasifikasi fimana masukan berupa prediksi merupakan nilai diantara 0 dan 1[32]. Tujuan dari *machine learning model* yang dibuat adalah untuk meminimalisir nilai ini. *Log loss* akan semakin bertambah apabila hasil prediksi jauh berbeda dengan label sebenarnya. Secara matematis, fungsi ini digambarkan dengan:

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

  dimana rumus diatas adalah penjumlahan dari seluruh nilai *log loss* pada setiap kelas dengan:
  M = jumlah kelas
  y = indikator biner (0 atau 1) dari apakah label *c* merupakan label yang benar untuk data observasi *o*
  p = probabilitas prediksi dari model dimana data observasi *o* masuk ke kelas *c*
  c = kelas, dimana nilai pertama adalah 1

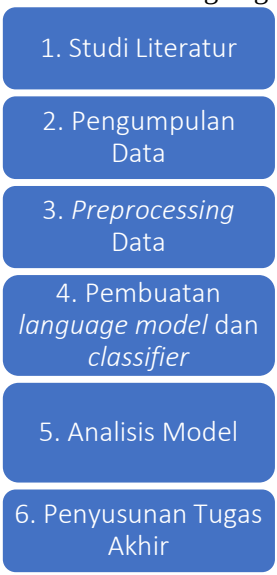
Halaman ini sengaja dikosongkan

BAB III
METODOLOGI

Bab ini menjelaskan tentang bagaimana metodologi yang akan digunakan pada tugas akhir. Metodologi ini akan menjadi panduan pada tugas akhir agar pengerjaannya dapat terarah dan sistematis.

3.1 Diagram Metodologi

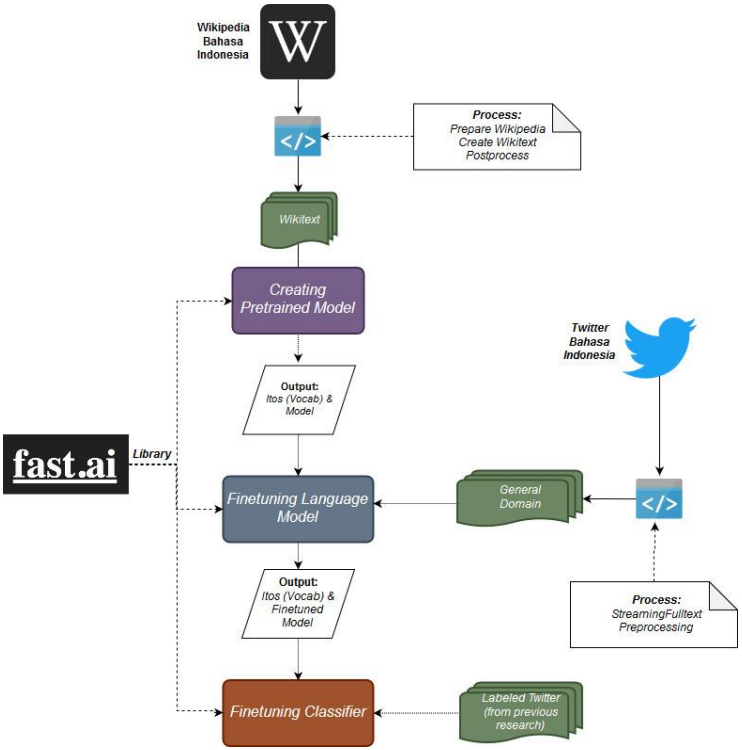
Pada sub bab ini akan dijelaskan mengenai tahapan yang dilakukan dalam penelitian sesuai dengan gambar 3.1.



Gambar 3.1 Metodologi Pengerjaan Tugas Akhir

3.2 Arsitektur Penelitian

Pada gambar 3.2, dijelaskan mengenai arsitektur penelitian dari tugas akhir ini.



Gambar 3.2 Arsitektur Penelitian

Pada gambar 3.2, dapat dilihat bahwa masukan awal dari proses merupakan data dari Wikipedia Bahasa Indonesia. Setelah mengalami proses pengunduhan serta pra-pemrosesan yang lain, data akan digunakan untuk proses *training* menggunakan *library* FastAi. Proses pembuatan *pretrained model* ini akan menghasilkan *file vocabulary* berupa *itos* serta model dengan ekstensi *pth*.

Pada proses selanjutnya, dilakukan *finetuning* pada *pretrained model* dengan menggunakan *language model* yang dibentuk dari data *twitter* berdomain *general* dan telah mengalami pra-pemrosesan. Proses ini dilakukan agar *pretrained model* mampu meningkatkan kualitas *classifier* yang nantinya dibentuk dari data berlabel meskipun hanya berjumlah sedikit.

Tahapan berikutnya adalah pembentukan *classifier* dengan menggunakan model yang telah mengalami proses *fine-tuning* dan *databunch* yang menyimpan data berlabel.

3.3 Uraian Metodologi

Sub bab ini akan menjelaskan mengenai uraian tahapan metodologi yang akan dijalankan pada pengerjaan tugas akhir.

3.2.1. Studi Literatur

Pada tahapan studi literatur akan dilaksanakan pemahaman konsep-konsep, metodologi, serta studi kasus, agar didapatkan pemahaman lebih lanjut mengenai kondisi dari studi kasus serta konsep dan metodologi dari solusi yang akan diterapkan. Rujukan utama yang digunakan sebagai acuan dari tugas akhir ini adalah *paper* dari Jeremy Howard dan Sebastian Ruder yaitu *Universal Language Model Fine-tuning for Text Classification*[8]. Selain itu,penulis juga merujuk pada *paper* berjudul *ULMFiT at GermEval-2018: A Deep Neural Language Model for the Classification of Hate Speech in German Tweets*[9] oleh Kristian Rother dan Achim Rettberg, untuk memberikan gambaran nyata mengenai implementasi ULMFiT sebagai metodologi dalam klasifikasi teks dari media sosial *Twitter*. *SemEval-2017 Task 4 : Sentiment Analysis in Twitter*[11] oleh Sara Rosenthal, Noura Farra, dan PreslaV Nakov juga digunakan sebagai panduan pelaksanaan analisis sentimen dalam tugas akhir, terutama dalam pemberian *subtask.*

3.2.2. Pengumpulan Data

Tahapan pengumpulan data dilaksanakan dengan mengambil data lewat proses *streaming*, dimana akan diambil data *tweet* berbahasa Indonesia yang menggunakan kata kunci tertentu yang berbahasa Indonesia pula secara *real time*. Data dari proses *streaming* ini akan digunakan sebagai data dalam pembuatan *language model.* Selain proses *streaming*, terdapat pula proses *crawling* pada *tweet* dari pengguna media sosial yang menyebutkan kata kunci yang berkaitan dengan *e-commerce*. Pada tabel 3.1, terdapat contoh dari *tweet* dengan *filter* yang akan dikumpulkan untuk proses pembuatan pengklasifikasi.

Tabel 3.1 Contoh *tweet* dengan kata kunci

Tweet	Filter
Flash sale di **shopee** aja **cepet** laku	Menyebutkan kata *shopee* yang merupakan salah satu objek yang diamati dalam studi kasus
@ShopeeID kak no Pesanan COD #19031512092DF8A sudah jalan dari tanggal 16 tapi kok status di shopee tidak berubah di tap DIKIRIM. mohon bantuanya **@ShopeeCare**	Melakukan *mention* ke akun *@ShopeeID* yang merupakan akun resmi *shopee* di *twitter*

Library yang digunakan pada Python untuk proses ini adalah *tweepy*. Selain melaksanakan *crawling* pada *twitter*, data diperoleh dari pengambilan data yang dilaksanakan pada periode sebelumnya, serta dengan mengambil data yang disediakan oleh Wikipedia Bahasa Indonesia.

3.2.3. *Preprocessing* Data

Pada sub bab ini, akan dijelaskan mengenai tahapan *preprocessing* data yang diimplementasikan pada tugas akhir.

3.2.4. Penyaringan Data

Proses ini dilakukan untuk penyaringan data yang mengandung kata-kata berbahasa asing dengan *library langdetect* dari Python. Sehingga nantinya, data yang diolah dalam tugas akhir merupakan data berbahasa Indonesia.

3.2.5. Pelabelan Data

Pelabelan data untuk hasil dari proses *streaming* dilakukan dengan mengganti *id* dari *tweet* menjadi angka 0, hal ini dikarenakan *tweet* yang berdomain *general* harus memiliki label 0 sesuai dengan format yang ditentukan oleh *FastAi.*

Selain itu, dilakukan pula pelabelan untuk data pengklasifikasi. Hanya saja, pelabelan ini telah dilakukan

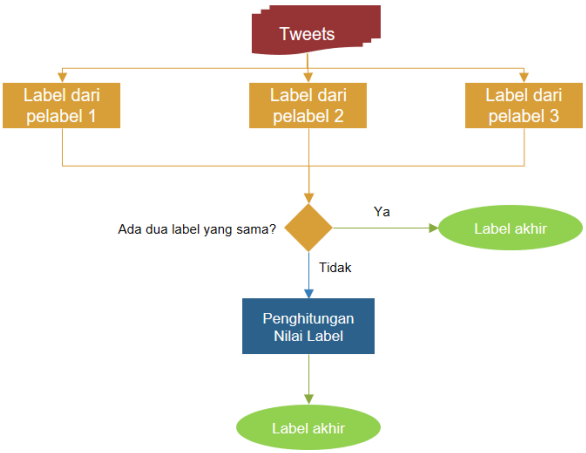
sebelumnya pada tugas akhir yang lalu. Pada proses ini, data yang telah disaring dan sesuai dengan topik diberi label oleh *annotator*. Dengan menggunakan acuan pada *SemEval-2017*[11], maka akan digunakan tiga dari lima *subtask* yang tercantum di dalamnya, yaitu:

a. Subtask A : Positif dan Negatif
b. Subtask B : Positif, Netral dan Negatif
c. Subtask C : Sangat Positif, Positif, Netral, Negatif, Sangat Negatif

Pelabelan data akan dilaksanakan oleh tiga *annotator*. Apabila dua dari tiga *annotator* setuju akan label yang diberikan, atau dalam kata lain memberikan label yang sama, maka label akan digunakan. Jika tidak terjadi kesepakatan, maka dilakukan konsolidasi untuk anotasi yang ada. Sebagaimana pada *SemEval-2017*[11], tiap kategori pada label akan diberikan nilai berupa integer. Berikut merupakan nilai yang diberikan berdasarkan label dari data:

- Sangat Negatif : -2
- Negatif : -1
- Netral : 0
- Positif : 1
- Sangat Positif : 2

Setelah diberikan nilai, maka akan dihitung rata-rata dari label yang diberikan oleh *annotator*. Kemudian nilai tersebut dibulatkan ke nilai integer terdekat untuk diberikan label. Nilai dengan angka desimal ≤ .4 akan dibulatkan ke integer terkecil, begitu juga sebaliknya. Setelah itu, data akan diberikan label sesuai nilai integer yang didapatkan. Gambaran dari proses pemberian label ini dapat dilihat pada gambar 3.2.



Gambar 3.3 Proses Pemberian Label pada Tweet

3.2.6. Pra-proses Data

Pra-proses data dilaksanakan agar data yang didapatkan dari pengumpulan data dapat dimanfaatkan dalam proses klasifikasi untuk menghasilkan akurasi yang lebih baik. Pra-proses yang dilaksanakan antara lain adalah *case folding*, yang bertujuan untuk menyamakan huruf pada data teks agar menjadi huruf kecil, penggantian *mention* dan *url* menjadi teks, penghapusan huruf yang berulang lebih dari dua kali, penghapusan angka dan tanda baca, penggantian emotikon menjadi kata yang mewakilinya. Pra-pemrosesan data pada dokumen dari Wikipedia Bahasa Indonesia dan dari *twitter* memiliki proses yang berbeda, dikarenakan pada artikel Wikipedia tidak adanya *mention, emoji*, dan *url*, yang ditemukan pada data dari *twitter*. Pra-pemrosesan dari data *Wikipedia* akan mengikuti aturan dari S. Merity. Hasil dari pra-proses data dari *twitter* adalah dokumen *corpus* yang dimanfaatkan untuk proses *fine-tuning* pada *language model* (untuk data dengan domain umum) serta pembuatan *classifier* untuk klasifikasi sentiment (untuk data dengan domain *e-commerce*). Sedangkan data dari Wikipedia dimanfaatkan pada tahapan *pre-training* pada *language model.*

3.2.7. Proses Pembuatan *Language Model* dan *Classifier*

Pada sub bab ini, akan dituliskan penjelasan mengenai pembuatan *language model* dan *classifier* menggunakan ULMFiT, mulai dari proses *pretraining* hingga *fine-tuning classifier.*

3.2.8. Proses *Pre-train Language Model*

Pada proses ini, dilakukan *pre-training* pada *language model* dengan menggunakan implementasi ULMFiT pada fast.ai. Model akan dilatih menggunakan data yang telah disediakan oleh *Wikipedia* Bahasa Indonesia. *Language model* yang dihasilkan diharapkan mampu untuk menggali fitur-fitur dari teks untuk tugas atau *task* turunan seperti klasifikasi teks.

3.2.4.1 Proses *Fine-tune Language Model*

Pada proses ini, dilakukan *fine-tune* pada *language model* yang telah dibentuk sebelumnya dengan menggunakan data dari *twitter* berupa *tweet* yang tidak berlabel, yang telah dibagi menjadi 20% data *testing*, 20% data *validation* dan 60% untuk *training*. Tujuan dari *fine-tuning* adalah agar *language model* dapat mempelajari fitu-fitur spesifik pada tugas.

3.2.4.2 Proses *Fine-tune Classifier*

Pada proses ini, dilakukan *fine-tune* pada *classifier* dengan menggunakan *language model* yang ada. Proses klasifikasi menggunakan tiga layer LSTM. *Fine-tuning* pada *classfier* bertujuan agar *classifier* dapat mempertahankan representasi level rendah dan beradaptasi dengan representasi level tinggi atau lebih spesifik dari data. Sumber data yang digunakan pada proses ini adalah *tweet* yang telah diberi label dan telah dipisahkan menjadi data *training* dan *testing*.

3.2.9. Analisis Model

Pada tahapan ini, dilakukan perhitungan terhadap performa yang diperoleh model. Untuk melakukan perhitungan performa, digunakan nilai akurasi yang juga digunakan pada penelitian terkait, serta digunakan pula nilai *F1-score* yang telah banyak digunakan dalam pengukuran akurasi dari tugas dalam NLP. Akurasi merupakan pengukuran untuk menilai seberapa baik pengklasifikasi, dimana nilai yang diukur adalah

hasil dari jumlah hasil klasifikasi yang benar (*true positives* dan *true negatives*) dibandingkan dengan jumlah seluruh kasus atau data yang diklasifikasi[33]. Sedangkan *F1-score* diartikan sebagai rata-rata harmonik dari presisi (*true positive* dibagi dengan hasil *positive*) dan *recall* (*true positive* dibagi dengan kesuluruhan data *positive*)[34]. Nilai dari akurasi dan *F1-score* sendiri nantinya akan diperoleh dari *library* sklearn.

3.2.10. Penyusunan Tugas Akhir

Pada tahapan ini dilaksanakan pendokumentasian hasil dari setiap metodologi yang dilaksanakan ke dalam bentuk buku tugas akhir, yang diharapkan dapat bermanfaat sebagai referensi pada penelitian berikutnya.

BAB IV
PERANCANGAN

Pada bab ini akan dijelaskan lebih detail mengenai metodologi yang akan digunakan dalam penyusunan tugas akhir. Metodologi ini kemudian akan digunakan sebagai panduan dalam perancangan serta penyusunan tugas akhir agar terarah dan sistematis.

4.1 Akuisisi Data Media Sosial

Tahapan awal dari perancangan pengerjaan tugas akhir ini dimulai dengan perencanaan akuisisi data dari media sosial, yang nantinya akan digunakan dalam proses pembuatan model bahasa dan pengklasifikasi. Media sosial yang akan menjadi sumber data teks pada tugas akhir ini adalah *Twitter*, dikarenakan banyaknya data teks yang beragam dan sesuai dengan *task* analisis sentimen, serta ketersediaan API yang mudah dimanfaatkan untuk bidang NLP.

Akuisisi data ini dibagi menjadi dua jenis, yang pertama adalah data untuk proses pembuatan *language model*, dan yang kedua untuk proses pembuatan pengklasifikasi. Data untuk proses pembuatan *language model* adalah data dengan domain umum, sehingga *language model* dapat menyimpan ciri-ciri kebahasaan dari teks pada media sosial *Twitter*. Berbeda dengan data untuk pengklasifikasi yang harus sesuai konteks, dimana dalam tugas akhir ini mengenai lima *e-commerce* besar yang ada di Indonesia.

Untuk melaksanakan hal tersebut, dibuatlah *streamer* yang akan mengambil data *tweet* berbahasa Indonesia dengan mengambil kata kunci yang mencirikan bahasa Indonesia. Pada tabel 4.1, ditunjukkan kata kunci yang digunakan untuk proses *streaming* media sosial beserta alasan digunakannya kata kunci tersebut.

Tabel 4.1 Contoh Kata Kunci Streaming

No.	Kata Kunci	Alasan Penggunaan
1.	Aku	Merupakan kata ganti orang pertama pada bahasa Indonesia dan seringkali dipakai dalam penulisan tweet

2.	Kamu	Merupakan kata subjek dalam bahasa Indonesia dan seringkali digunakan dalam tweet
3.	Gue	Merupakan kata ganti orang pertama dalam bahasa Indonesia, namun tidak baku, sehingga dapat menangkap gaya bahasa yang beragam dalam tweet
4.	Nggak	Merupakan kata dalam bahasa Indonesia yang menyatakan penyangkalan, penolakan, dan sejenisnya, namun bersifat tidak baku, sehingga dapat menangkap gaya bahasa yang beragam dari tweet
5.	Mantab	Merupakan kata yang menunjukkan kepuasan dalam bahasa Indonesia, sehingga dapat menambah kosakata dari tweet

Untuk membandingkan performa dari pengklasifikasi, maka digunakan data dari proses *crawling* yang diperoleh dari tugas akhir periode 2018. Pada *crawler* yang digunakan dalam proses penggalian data tersebut, digunakan kata kunci yang mencerminkan topik dari analisis sentimen, yaitu *e-commerce*. Pada tabel 4.2, dijelaskan mengenai kata kunci yang digunakan pada proses *crawling* pada tugas akhir periode 2018 lalu.

Tabel 4.2 Kata Kunci Pada Dataset *Twitter* Bertopik

Kata Kunci
tokopedia
tokopediacare
bukalapak
bukabantuan

bliblidotcom
bliblicare
lazadaid
lazadacare
shopee
shopeecare

4.2 Perancangan Penggalian Data Media Sosial

Streamer yang pada tugas akhir ini berfungsi untuk memperoleh tweet yang bersifat *realtime* dan mengandung kata kunci untuk language model yang telah disebutkan sebelumnya. API yang digunakan pada *streamer* berasal dari *twitter*. Selain itu, digunakan pula *library tweepy* dan bahasa pemrograman *python*. *Streamer* akan mengambil data *tweet* yang sesuai dengan kata kunci yang telah disebutkan dalam kurun waktu satu minggu. Data yang dihasilkan dalam kode pada *streamer* adalah data dengan format *json*.

4.3 Perancangan *Pre-processing* Data Media Sosial

Pada sub bab ini akan dijelaskan mengenai perancangan dari prapemrosesan yang akan diterapkan terhadap data *tweet.*

4.3.1 Perancangan Ekstraksi Tweet

Hasil akhir dari proses *streaming* pada tahapan sebelumnya adalah suatu file dengan ekstensi *txt* yang berisi atribut dari tiap *tweet* berformat *json*. Untuk dapat melanjutkan pada proses berikutnya, maka harus diperoleh atribut dari *tweet* yang telah didapatkan. Proses ekstraksi ini dapat dilakukan dengan menggunakan kode *python*. Setelah memperoleh atribut dari tiap *tweet*, maka *tweet* dapat disimpan ke dalam suatu file csv untuk dapat dimanfaatkan pada proses berikutnya.

Selain pengambilan data teks dari *tweet* saja, dilakukan pula pemberian kelas kepada *tweet* untuk *language model*, yaitu kelas nol, untuk menyesuaikan dengan *library* FastAi.

Sedangkan untuk data pengklasifikasi, data telah disesuaikan dengan format *txt* dan mengandung atribut-atribut yang diperlukan, sehingga tidak memerlukan proses ekstraksi lebih lanjut.

4.3.2 Perancangan Filtering Bahasa

Dalam proses *streaming*, dapat diperoleh *tweet* dengan menggunakan kata kunci berbahasa Indonesia yang telah dicantumkan. Meskipun begitu, terdapat kemungkinan bahwa *tweet* yang diperoleh menggunakan bahasa lain selain bahasa Indonesia. Hal ini bisa terjadi apabila kata kunci juga merupakan kata dalam bahasa lain, ataupun *tweet* tersebut merupakan campuran dari bahasa Indonesia dengan bahasa lain. Untuk mengatasi persoalan ini, maka dilakukan penyaringan bahasa yang terdapat dalam *tweet* dengan menggunakan *library langdetect* pada *python*. Dengan menggunakan *langdetect*, dapat diperoleh indeks probabilitas dari tiap-tiap bahasa yang mungkin dimiliki teks pada *tweet*. Pada tugas akhir ini, akan diambil teks dengan probabilitas bahasa Indonesia minimal 0.5, dan teks lain yang memiliki probabilitas lebih kecil dari 0.5 akan dihapus atau tidak digunakan. Sedangkan pada data hasil proses klasifikasi, telah dilakukan proses penyaringan bahasa sehingga telah siap untuk dimanfaatkan pada proses selanjutnya.

4.3.3 Perancangan Pembersihan Data

Data yang telah diperoleh dari proses *streaming* memiliki berbagai karakter yang kurang berguna pada proses *training*, sehingga pada tahapan ini, karakter tersebut dihapus. Selain itu, terdapat pula hashtag dan mention yang beragam dari tiap tweet yang akan disederhanakan agar proses *training* dapat berjalan lebih efektif.

Tabel 4.3 Proses Pembersihan Data

No.	Proses
1.	Menghapus *url* yang terdapat pada teks
3.	Mengganti *hashtag* (misal, #Diskon) pada teks *dengan string* 'hashtag'

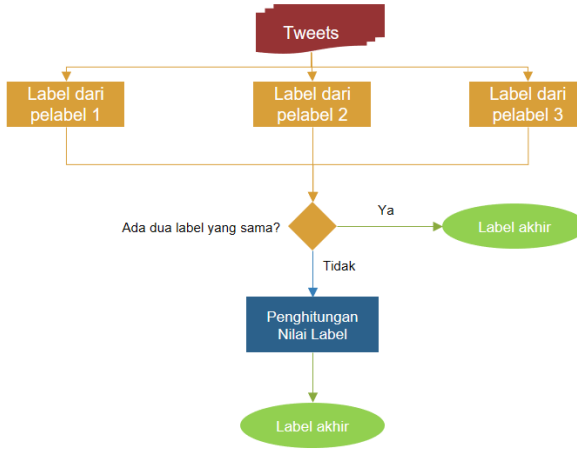
4.	Mengganti *mention* (misal, @BlibliCare) pada teks *dengan string* 'mention'
5.	Mengganti huruf atau karakter yang muncul lebih dari dua kali, menjadi maksimal hanya 2 kali. Misal, 'haloooo' menjadi 'haloo'
6.	Mengganti emotikon menjadi teks yang mewakili emotikon tersebut. Misal ':D' menjadi 'tertawa'
7.	Menghapus karakter selain huruf
8.	Melakukan *casefolding*, dimana huruf kapital diubah menjadi huruf kecil

Data yang diperoleh dari proses ini selanjutnya akan diproses menggunakan *MosesTokenizer* pada tahapan berikutnya sebelum dimanfaatkan oleh *language model*.

4.3.4 Pra-pemrosesan dan Pelabelan Data Pengklasifikasi

Pada data pengklasifikasi, setelah diperoleh dan dilakukan ekstraksi, tahapan setelahnya adalah melakukan pelabelan sesuai dengan *subtask* yang akan diteliti. Di bab sebelumnya, telah dijelaskan bahwa ada tiga jenis *subtask* yang akan digunakan, dengan jumlah jenis label maksimal adalah lima label.

Data yang diperoleh pada penelitian ini telah mengalami proses pelabelan. Proses pertama yang dilakukan adalah tiga orang pelabel memberikan label pada masing-masing label dengan kelas label berupa sangat positif, positif, netral, negatif, dan sangat negatif. Label-label tersebut disimbolkan dengan angka (2 untuk sangat positif hingga -2 untuk sangat negatif). Setelah diperoleh label dari masing-masing tweet, dilakukan penghitungan rata-rata dari label masing-masing tweet. Untuk pembulatan, jika desimal kurang dari sama dengan 0.4, maka dilakukan pembulatan kebawah. Begitu juga sebaliknya. Proses pelabelan ini telah digambarkan pada gambar 4.1.



Gambar 4.1 Proses Pelabelan Data

Setelah dilakukan proses tersebut, maka diperoleh hasil akhir berupa data beserta label yang dituliskan pada tabel 4.4. Berdasarkan pengerjaan tugas akhir sebelumnya, diperoleh proporsi antar pelabel yang berupa *analisis Cohen's Kappa* dengan skor rata-rata 0.348, dimana dapat ditarik kesimpulan bahwa tingkat persetujuan tersebut bernilai sedang.

Tabel 4.4 Jumlah Data per Label

Label	Jumlah Data
Sangat Positif	33
Positif	1493
Netral	7151
Negatif	1553
Sangat Negatif	456
Total Data	10,686

Data yang telah berlabel kemudian diproses dengan perlakuan tertentu agar dapat memenuhi kebutuhan tiap *subtask* seperti yang dicantumkan pada tabel 4.5. Hasil dari pengolahan

tersebut adalah data yang telah berlabel sesuai dengan tiga *subtask* yang dibutuhkan.

Tabel 4.5 Perlakuan Label Pada Subtask

Subtask	Label	Perlakuan
A	Positif dan negatif	Menghapus data dengan label netral, mengubah label sangat negatif menjadi label negatif, mengubah label sangat positif menjadi label positif
B	Positif, netral, negatif	Mengubah label sangat negatif menjadi label negatif, mengubah label sangat positif menjadi label positif
C	Sangat positif, positif, netral, negatif, sangat negatif	Tidak mengalami perlakuan khusus

Data pengklasifikasi untuk selanjutnya mengalami proses tokenisasi dengan menggunakan *MosesTokenizer* sehingga dapat dilakukan pemrosesan lebih lanjut pada library FastAi.

4.4 Akuisisi Data Wikipedia

Akuisisi data dari suatu kumpulan data teks yang memiliki domain umum diperlukan. Hal ini dilaksanakan agar pada pretrained model, dapat diperoleh ciri-ciri dari suatu bahasa, termasuk pula kumpulan katanya. Karena alasan ini, maka dilaksanakanlah proses akuisisi data pada *Wikipedia* bahasa Indonesia. Tahapan ini dilakukan sesuai dengan praktik yang dilaksanakan pada *paper* dari ULMFiT[8]. Proses akuisisi dilakukan dengan memanfaatkan suatu *script* untuk proses pengunduhan *Wikipedia dumps* dengan bahasa Indonesia, serta melakukan *cloning* pada kode *WikiExtractor* yang telah tersedia untuk umum untuk proses ekstraksi pada tahapan selanjutnya.

4.5 Perancangan *Pre-processing* Data Wikipedia

Agar dapat dimanfaatkan oleh proses berikutnya, maka data dari *Wikipedia* melalui tahapan ekstraksi dan pra-pemrosesan. Data dari *Wikipedia* yang terkompresi akan diekstraksi, kemudian dibersihkan dari template dan karakter-karakter yang tidak dibutuhkan. Proses ekstraksi dijalankan oleh *WikiExtractor* yang menerima masukkan berupa *Wikipedia dump*, dan kemudian menghasilkan beberapa dokumen berupa elemen XML *doc* pada direktori yang telah ditentukan. *WikiExtractor* sendiri merupakan suatu skrip yang berfungsi untuk melakukan ekstraksi dan permbersihan pada *file Wikipedia dump* serta menyimpan luarannya menjadi beberapa file dengan ukuran yang seragam pada direktori tertentu.

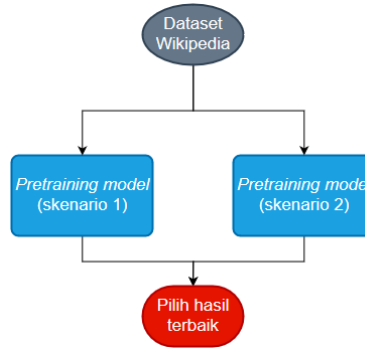
Setelah dilakukan ekstraksi, data dari *Wikipedia* tersebut masih berupa dokumen XML. Dokumen XML kemudian akan ditokenisasi dengan menggunakan *MosesTokenizer*, lalu diubah menjadi tiga jenis kumpulan token, yaitu token untuk *train*, *test*, dan *validation* pada *pretrained model*.

Kumpulan token yang telah disimpan sebagai suatu file kemudian akan mengalami *post-process*, dimana akan dilakukan penggantian angka menjadi suatu *placeholder*, pembuatan *vocabulary*, serta pembatasan ukuran *vocabulary*.

4.6 Perancangan Model ULMFiT

4.6.1 Perancangan *Pretrained Model* ULMFiT

Dalam pembuatan model ULMFiT, akan digunakan library FastAi versi 1.0.27. FastAi merupakan salah satu *library* yang digunakan dalam *machine learning*. Penggunaan FastAi versi 1.0.27 dikarenakan masih adanya dukungan *pretrained model* pada bahasa lain selain bahasa Inggris, namun tetap tergolong baru sehingga dapat dikomparasi dengan versi terkini. Ringkasan mengenai proses pelaksanaan *pretraining model* dijelaskan pada gambar 4.2.



Gambar 4.2 Pelaksanaan *pretraining model*

4.6.1.1 Perancangan Dataset *Pretrained Model*

Pada tahapan ini, data dipersiapkan menjadi kumpulan token yang terdiri dari *train*, *test*, dan *validation dataset*. *Dataset* yang digunakan merupakan data *Wikipedia* yang telah mengalami *post-process*.

Selain menggunakan data dari *Wikipedia*, pada tugas akhir ini juga dilakukan eksperimen dengan memanfaatkan hasil *streaming* dari media sosial *Twitter* sebagai data untuk model *pretrained*. Sebelum dilakukan proses *training*, data akan disimpan dalam suatu *DataBunch* yang berfungsi untuk menyimpan dataset yang akan diproses menggunakan *library* FastAi.

4.6.1.2 Perancangan Pretrained Model

Proses pembuatan *pretrained model* bertujuan agar model mampu memahami fitur-fitur kebahasaan umum dari data yang akan diklasifikasi. Proses ini seperti pada proses lainnya memanfaatkan *library* dari FastAi. Pembuatan model dilakukan dengan membuat objek *language model learner* menggunakan *databunch* yang telah dibuat pada tahapan sebelumnya. Beberapa parameter yang akan digunakan dalam proses pembuatan objek *language model learner* disebutkan dalam tabel 4.6. Sesuai dengan percobaan yang dilakukan oleh Howard dan Ruder, arsitektur yang digunakan dalam model ini adalah AWD LSTM dengan menggunakan ukuran embedding (*emb_sz)* 400, jumlah layer 3 (*nl*), dan jumlah aktivasi per layer (*nh*) sejumlah 1150.

Tabel 4.6 Parameter pada objek *language model learner*

No.	Parameter	Penjelasan
1.	*num_epochs*	*Number of epoch*, merupakan jumlah iterasi yang dilakukan pada proses training
2.	*dps*	*Dropouts*, merupakan dropout yang digunakan pada proses training untuk mengurangi overfitting
3.	*drop_mult*	*Dropouts multiplier*, sebagai pengali untuk mengubah ukuran dropout agar menyesuaikan ukuran probabilitas dropout pada arsitektur yang digunakan
4.	*opt_fn*	*Optimization function*, fungsi optimasi yang digunakan pada model
5.	*BPTT*	*Backpropagation through time*, ukuran panjang dari batch dokumen yang digunakan oleh BPTT
6.	*pad_token*	*Padding token*, indeks dari token yang digunakan untuk padding, secara default bernilai 1
7.	*clip*	*Gradient clip*, berfungsi untuk menerapkan nilai maksimal dari gradient clipping
8.	*true_wd*	*True weight decay*, berisi boolean yang jika bernilai True akan menerapkan *Fixing Weight Decay Regularization in Adam*[35], berlaku untuk semua optimalisasi

Setelah pembuatan objek *language model learner* dilangsungkan, maka langkah selanjutnya adalah pelaksanaan

training dengan menggunakan fungsi *fit.* Parameter yang digunakan pada fungsi ini terdapat pada tabel 4.7. Objek yang telah mengalami proses *training* selanjutnya disimpan dan dijadikan sebagai objek *pretrained model* pada tahapan selanjutnya.

Tabel 4.7 Parameter proses *fitting* model

No.	Parameter	Penjelasan
1.	*num_epochs*	Number of epoch, merupakan jumlah iterasi yang dilakukan pada proses training
2.	*lr*	Learning rate, nilai ukuran pembelajaran yang digunakan pada proses training
3.	*WD*	Weight decay, faktor pengali pada bobot yang bernilai kurang dari 1 untuk mencegah bobot berkembang terlalu besar
4.	*lrs*	Array yang berisi pembagi dari learning rates untuk menerapkan aturan discriminative fine-tuning
5.	*div_factor*	Faktor pembagi learning rates maksimal untuk penerapan aturan 1cycle
6.	*pct_start*	Percentage start, berupa persentase dari cycle yang ingin digunakan pada awal, digunakan untuk penerapan aturan 1cycle
7.	*moms*	*Momentum*, berisi dua nilai yang mendefinisikan momentum maksimal dan momentum minimal untuk proses training, digunakan untuk penerapan aturan 1cycle

4.6.2 Perancangan *Finetune* ULMFiT

Meskipun telah menggunakan model *pretrained*, data dari task target bisa memiliki distribusi yang berbeda dengan model tersebut. Karena itu dilakukanlah proses *finetuning* dengan menggunakan target *task* agar *pretrained model* dapat beradaptasi dengan *idiosyncrasies* dari target task.

4.6.2.1 Perancangan Dataset *Language Model*

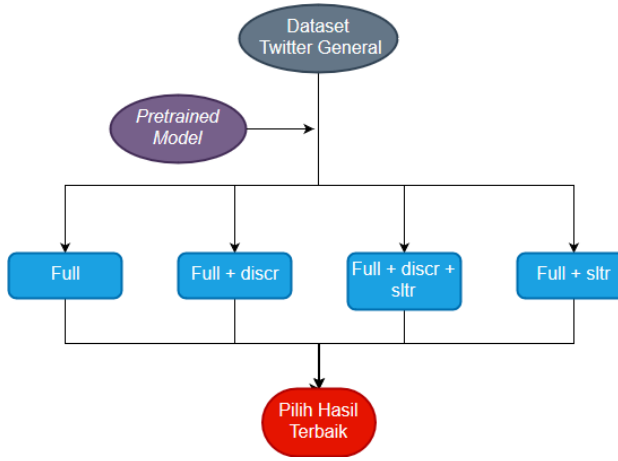
Dataset yang digunakan pada proses pembuatan model bahasa untuk fine-tuning adalah dataset dari twitter yang memiliki domain atau topik umum, sehingga pretrained model dapat menyesuaikan dengan *idiosyncrasies* pada jenis teks *Tweet* secara umum. Data tersebut dibagi menjadi data *train*, *test*, dan *validation*. Ukuran dari data *test* dan *validation* adalah 5% dari total data, sisanya adalah data *training*. Untuk selanjutnya, data akan disimpan menjadi objek *TextLMDataBunch* agar dapat diproses ke dalam tahapan berikutnya.

4.6.2.2 Perancangan *Fine-tuning Language Model*

Pada proses fine-tuning, dibuat sebuah objek *language model learner* yang nantinya akan membuat objek *Learner* (*trainer* untuk data) dari data pada dataset language model. Salah satu parameter dalam pembuatan objek language model learner adalah pretrained model, dimana akan dimasukan pretrained model yang telah dibuat pada tahapan sebelumnya. Selain itu, beberapa parameter lain yang digunakan pada objek ini masih sama dengan proses sebelumnya, yaitu ukuran embedding (emb_sz) 400, jumlah layer 3 (nl), dan jumlah aktivasi per layer (nh) sejumlah 1150. Setelah pembuatan objek language model learner, akan dilakukan proses training dengan memanfaatkan fungsi fit pada FastAi. Pada tabel 4.5, ditunjukkan beberapa parameter untuk proses fit pada language model learner.

Pada tahapan ini, akan dilakukan fine-tuning pada seluruh layer (*full*). Selain itu, terdapat beberapa metode lain yang dimanfaatkan dalam proses ini, diantaranya adalah metode *discriminative fine-tuning* (*discr*), dimana setiap *layer* akan menggunakan *learning rate* yang berbeda saat proses fine-

tuning. Selain itu, juga dimanfaatkan *slanted triangular learning rates* (*sltr*) dimana akan dilakukan peningkatan *learning rates* secara linear, kemudian secara linear pula menurunkan learning rates seiring dengan fase yang ditentukan. Terdapat empat skenario berbeda untuk proses *fine-tuning*, kemudian hasil fine-tuning terbaik akan digunakan pada tahapan selanjutnya. Gambaran mengenai proses ini telah dicantumkan pada gambar 4.3.



Gambar 4.3 Skenario *Fine-tuning Language Model*

4.6.3 Perancangan Classifier ULMFiT

4.6.3.1 Perancangan Dataset Pengklasifikasi

Pengklasifikasi menggunakan dataset yang berasal dari data tweet dengan topik *e-commerce* dan telah mengalami proses pelabelan sentimen dan telah disesuaikan dengan kebutuhan *subtask* masing-masing. Data teks berlabel kemudian akan dibagi menjadi data untuk *training*, *testing*, dan *validation*. Ukuran dari data *test* dan *validation* adalah 5% dari total data. Tahapan berikutnya adalah menyimpan data pengklasifikasi menjadi objek *TextClasDataBunch*. Objek *TextClasDataBunch* diatur agar memiliki *vocab* yang sama dengan *TextLMDataBunch* dengan memasukkan parameter *vocab* dari *language model*.

4.6.3.2 Perancangan Pembuatan Pengklasifikasi

Pada proses pembuatan pengklasifikasi, objek berupa TextClasDataBunch akan mengalami proses training dengan memanfaatkan language model yang dibuat sebelumnya serta membandingkannya dengan classifier tanpa language model. Objek *TextClasDataBunch* akan digunakan pada pembuatan classifier berupa objek *text_classifier_learner*. *Learner* yang digunakan pada proses klasifikasi ini memiliki beberapa parameter yang serupa dengan *language_model_learner*.

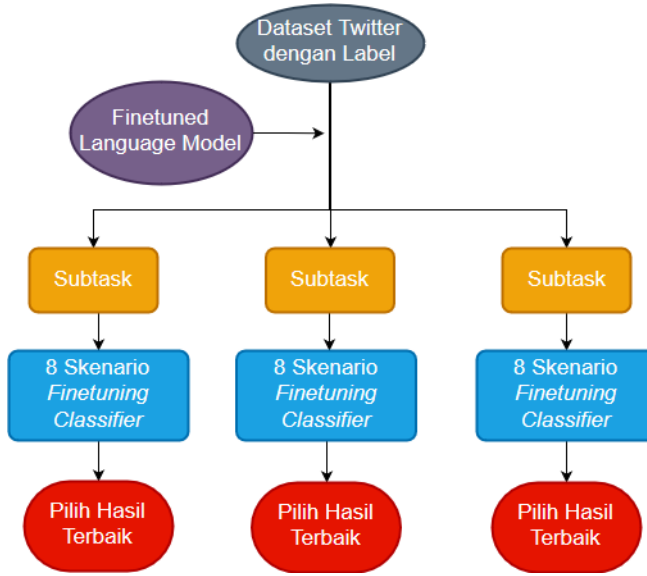
Pada tahapan ini, akan dimanfaatkan metode gradual unfreezing, dimana akan dilakukan proses fine-tuning pada classifier secara bertahap, diawali dengan layer terakhir. Metode ini akan dipadukan dengan metode lain berupa *slanted triangular learning rates* dan *discriminative learning rates*. Secara total, ada 8 skenario *fine-tuning classifier* yang akan diimplementasikan pada masing-masing *subtask*. Tabel 4.8 menerangkan skenario apa saja yang akan digunakan pada proses *fine-tuning classifier*.

Tabel 4.8 Skenario fine-tuning classifier

No.	Skenario	Penjelasan
1	*Full*	Dilakukan *finetuning* pada seluruh *layer classifier*
2	*Full + discr*	Skenario full dan digunakan pula discriminative learning rates
3	*Last*	Dilakukan *finetuning* pada *layer* terakhir *classifier*
4	*Freez*	Dilakukan *finetuning* secara bertahap pada tiap layer, menggunakan *gradual unfreezing*
5	*Freez + discr*	Skenario *freez* dengan memanfaatkan *discriminative learning rates*
6	*Freez + stlr*	Skenario *freez* dengan memanfaatkan *slanted triangular learning rates*

7	*Freez + cos*	Skenario *freez* dengan menggunakan *aggresife cosine annealing schedule*
8	*Freez + discr + sltr*	Skenario *freez* dengan memanfaatkan *discr* dan *sltr*

Secara rinci, gambaran mengenai tahapan ini telah dicantumkan pada gambar 4.4.



Gambar 4.4 Skenario *fine-tuning* tiap *classifier*

4.7 Perancangan Proses Evaluasi Pengklasifikasi

Proses evaluasi akan dilaksanakan dengan melakukan klasifikasi pada data *testing*, yang kemudian hasilnya akan diambil dan digunakan untuk penghitungan *precision*, *recall*, *accuracy*, dan *F1* menggunakan *library* sklearn.

Halaman ini sengaja dikosongkan

BAB V
IMPLEMENTASI

Bab ini akan menjelaskan mengenai implementasi dari perancangan dan metodologi yang telah dibahas pada bagian sebelumnya.

5.1 Lingkungan Implementasi

Tugas akhir ini dilakukan dengan menggunakan komputer dengan spesifikasi sebagai berikut:

Tabel 5.1 Spesifikasi Komputer

Prosesor	Intel i5 8400
Memory	16 GB DDR4 *Memory*
GPU	GTX 1070 8 GB
OS	*Linux Ubuntu* 16.04
Arsitektur	64 Bit

Selain itu, terdapat pula beberapa library, dan bahasa pemrograman yang membantu dalam pelaksanaan tugas akhir ini:

Tabel 5.2 *Library* pada Tugas Akhir

Bahasa Pemrograman	Python 3.7, Javascript
Editor	Sublime
Library	FastAi (v1.0.27) Fire Torch Pickle Sklearn Matplotlib Pandas Numpy Json

	Functools Pathlib Itertools

5.2 Pelaksanaan Akuisisi Data Media Sosial

Pelaksanaan akuisisi data media sosial dilakukan dengan proses *streaming* menggunakan API dari *Twitter* dan *library Tweepy*. Pada sub bab ini, akan dijelaskan mengenai proses *streaming* untuk mendapatkan data *twitter* dengan domain umum.

```
1.  consumer_key = 'XtoeL4Jz1FsccnV9GdlQIBAP5'
2.  consumer_secret = 'snlwbsd26KXQeYsQqxmQCttRzPdtqjIOmnUAIuTsbR7CC6N
    S8Y'
3.  access_token = '986217059418750976-
    BF15sJaOHRyOBCw7jyPkZ3EG8ipj165'
4.  access_secret = 'Xtjn2UKB0VXUAxlT5AzR0pT7QA0pNwY6yKQQX3Cgvb2Ws'
5.
6.  auth = OAuthHandler(consumer_key, consumer_secret)
7.  auth.set_access_token(access_token, access_secret)
8.
9.  tweets = []
10. filename = str(datetime.datetime.now().strftime("%Y%m%d %H%M%S"))
11.
12. save_file = open('data_'+filename+'.txt', 'a', errors="ignore")
13. print("file saved: data_"+filename+'')
14.
15. api = API(auth)
16.
17. try:
18.     tweetCount
19. except NameError:
20.     tweetCount = 0
```

Kode 5.1 Pendefinisian Akses API dan nama *file*

Potongan kode 5.1 menggambarkan proses autorisasi pada API Twitter untuk dapat melakukan *streaming*. Proses autorisasi tersebut digambarkan pada pendefinisian objek OAuthHandler() yang membutuhkan *consumer_key* dan *consumer_secret*. Setelah itu, hal yang dilakukan adalah memasukkan *access_token* dan *access_secret* yang diperoleh

dari aplikasi kita yang telah didaftarkan di halaman *developer* Twitter.

Selain menggambarkan proses autorisasi API, pada potongan kode 5.1 juga terdapat proses pendefinisian nama *file* dan penyimpanan *file* hasil *streaming*. *TweetCount* sendiri merupakan variabel yang digunakan untuk menyimpan jumlah dari *tweet* yang telah diambil lewat proses *streaming.*

```
1.   def on_data(self, data):
2.       all_data = json.loads(data)
3.       if 'extended_tweet' in all_data:
4.           fulltext = all_data['extended_tweet']['full_text']
5.       else:
6.           fulltext = ''
7.       if 'text' in all_data:
8.           tweet_text = all_data['text']
9.       else:
10.          tweet_text = ''
11.      if 'id_str' in all_data:
12.          tid = all_data['id_str']
13.      elif 'id' in all_data:
14.          tid = all_data['id']+''
15.      else:
16.          tid = ''
17.      text = {'id': tid, 'text': tweet_text, 'fulltext': fulltext}
18.      save_file.write(str(text)+"\n")
19.      self.counter+=1
20.      global tweetCount
21.      tweetCount = self.counter
22.      if self.counter % 50 == 0:
23.          print("saved tweets: "+str(self.counter))
24.      return True
```

Kode 5.2 Kelas *MyListener* Pada *Streamer*

Potongan kode 5.2 menunjukkan kode pada kelas MyListener() dimana *streamer* dibuat. Pada kode tersebut, ditunjukkan bahwa *streamer* akan menyimpan data pada variable *save_file* yang didefinisikan pada fungsi *__init__*. Kemudian setiap data masuknya, pada fungsi *on_data*, *streamer* akan melakukan *load* pada data yang berformat *json*, melakukan *append* ke *list save_file*, lalu *print* keterangan berapa data yang telah didapat pada kelipatan data 50. Data yang hanya akan

disimpan adalah data berupa *text* (apabila *tweet* hanya dituliskan secara singat), *fulltext* (apabila *tweet* memiliki *tweet* yang dapat diperpanjang), dan id.

```
1.  while True:
2.      try:
3.          twitterStream = Stream(auth, MyListener(),
    tweet_mode='extended')
4.          twitterStream.filter(track=["aku, kamu, gue, nggak, mantab
    , anjir, adalah, suatu"])
5.      except SSL.WantReadError as e:
6.          print("SSL WantReadError, sleeping...")
7.          time.sleep(5)
8.      except (ProtocolError, ConnectionError, ReadTimeoutError, Time
    out) as e:
9.          print("Network error happened, sleeping...")
10.         print(str(e))
11.         time.sleep(5)
12.     except KeyboardInterrupt:
13.         print("System exit, file data_"+filename+" saved...")

14.         sys.exit(1)
```

Kode 5.3 Proses Streaming dan Eksepsi

Pada potongan kode 5.3, proses *streaming* dijalankan selama kondisi bernilai *true. Tweet_mode* merupakan suatu mode agar *tweet* yang diambil dapat dibaca secara lengkap apabila melewati batas karakter yang telah ditentukan. Objek *Stream* yang disimpan pada variabel *twitterStream* akan melakukan *tracking* pada kata-kata yang dimasukkan diparameter *track.* Jika terjadi *error* berupa gangguan internet, *streamer* akan melakukan *pause,* kemudian melanjutkan *streaming* apabila koneksi kembali stabil.

5.3 Pelaksanaan *Pre-processing Dataset* Media Sosial

Pelaksanaan proses *pre-processing* dilakukan pada data hasil *streaming* berupa *tweet* dengan domain umum. Pada sub bab ini akan dijelakan mengenai kode-kode yang dijalankan pada proses *pre-processing dataset.* Seluruh kode pada sub bab 5.3.1-5.3.3 dirangkum dalam satu kelas, sedangkan 5.3.4 pada *file* yang lain.

5.3.1 Ekstraksi *Tweet*

Hasil pada proses *streaming* merupakan *file* yang memiliki ekstensi *txt* berisi *dictionary* data *tweet.* Agar tiap *value* dari *dictionary* tersebut dapat diakses, maka isi dari *file* tersebut akan dikonversi kedalam format json.

```
1.  class Preprocessing():
2.
3.      def __init__(self,
4.          filenames = ['data_20190629 212246.txt'],
5.          ):
6.          self.filenames = filenames
7.
8.      def combine(self):
9.          exist = os.path.isfile("result1.txt")
10.         if exist:
11.             print("file1 already exist, skipping combine...")
12.         else:
13.             with open('result1.txt','w', encoding="utf8") as ou
    tfile:
14.                 for fname in tqdm(self.filenames):
15.                     with open(fname, encoding="utf8", errors='i
    gnore') as infile:
16.                         for line in infile:
17.                             outfile.write(line)
18.
19.     def into_json(self):
20.         exist = os.path.isfile("result2.json")
21.         if exist:
22.             print("file2 already exist, skipping combining.")
23.         else:
24.             data=[]
25.             with open('result1.txt','r', encoding="utf8", error
    s='ignore') as file:
26.                 for line in tqdm(file):
27.                     line = line.replace("\n"," ")
28.                     line = ast.literal_eval(line)
29.                     data.append(line)
30.             with open('result2.json', 'w') as outfile:
31.                 json.dump(data, outfile)
```

Kode 5.4 Pengekstraksi Teks Tweet

Pada kode 5.4, terdapat dua fungsi yang ditampilkan dalam kelas *Preprocessing*. Fungsi pertama adalah *combine* yang akan menyimpan beberapa *file* hasil *streaming* kedalam satu *file.*

Fungsi kedua pada potongan kode 5.4. adalah fungsi into_json, dimana *file* akan disimpan ke dalam format json agar tiap *values* dari *dictionary tweet* dapat diakses.

5.3.2 Penyaringan Bahasa

```
1.  def filter_lang(self):
2.      print("Get only id language")
3.      id_data=[]
4.      nonidCounter=0
5.      with open('result2.json') as file:
6.          file = json.load(file)
7.          for line in tqdm(file):
8.              try:
9.                  lang = str(detect_langs(line['text']))[1:3]
10.                 probability = str(detect_langs(line['text']))
11.                 probability = float(probability[4:16])
12.                 if lang=="id" and probability>0.5:
13.                     id_data.append({
14.                         'id': line['id'],
15.                         'text': line['text']
16.                     })
17.                 else:
18.                     nonidCounter+=1
19.             except:
20.                 nonidCounter+=1
21.         print("Indonesia : "+str(len(id_data)))
22.         print("Non-Indonesia : "+str(nonidCounter))
23.     return id_data
```

Kode 5.5 Penyaringan Data Berbahasa Indonesia

Potongan kode 5.5 merupakan salah satu fungsi dari kelas *Preprocessing* yang digunakan sebagai penyaring bahasa pada *tweet.* Fungsi *filter_lang* akan membaca data dari *file json* yang telah dihasilkan pada tahapan sebelumnya. Menggunakan *langdetect*, akan dilakukan pendeteksian dari bahasa yang digunakan pada tiap *text* di objek *tweet*. Jika bahasa yang digunakan merupakan bahasa Indonesia, serta probabilitasnya

lebih dari 0.5, maka teks tersebut akan disimpan dalam *list* baru untuk kemudian diproses pada tahapan selanjutnya.

5.3.3 Pembersihan Data

Tahapan pembersihan data dilakukan dengan menghapus karakter yang tidak diperlukan serta mengganti emotikon sesuai dengan ekspresi yang diwakili oleh emotikon tersebut.

```
def replace_emoticons(string):
        # Campur
        string = string.replace('<3', ' <hati> ')
        string = string.replace(':D', ' <tertawa> ')
        string = string.replace(':P', ' <menjulurkan_lidah> ')
        string = string.replace(':p', ' <menjulurkan_lidah> ')
        string = string.replace(':o', ' <terkejut> ')
        string = string.replace(':O', ' <terkejut> ')
        string = string.replace(':x', ' <cium> ')
        string = string.replace(':*', ' <cium> ')
        string = string.replace(':3', ' <malumalu_kucing> ')
        string =string.replace('XD','<tertawa_terbahak-bahak>')

        # Senyum
        string = string.replace(':))', ' <senyum_senyum> ')
        string = string.replace(':)', ' <senyum> ')
        string = string.replace(':-))', ' <senyum_senyum> ')
        string = string.replace(':-)', ' <senyum> ')
        string = string.replace('((:', ' <senyum_senyum> ')
        string = string.replace('(:', ' <senyum> ')
        string = string.replace('((-:', ' <senyum_senyum> ')
        string = string.replace('(-:', ' <senyum> ')
        string = string.replace('=))', ' <senyum_senyum> ')
        string = string.replace('=)', ' <senyum> ')
        string = string.replace('^_^', ' <senyum> ')

        # Sedih
        string = string.replace(':((', ' <sedih_sedih> ')
        string = string.replace(':(', ' <sedih> ')
        string = string.replace(':-((', ' <sedih_sedih> ')
        string = string.replace(':-(', ' <sedih> ')
        string = string.replace(')):', ' <sedih_sedih> ')
        string = string.replace('):', ' <sedih> ')
        string = string.replace('))-:', ' <sedih_sedih> ')
        string = string.replace(')-:', ' <sedih> ')

        # Berkedip
```

```
        string = string.replace(';))', ' <senyum_berkedip> ')
        string = string.replace(';)', ' <senyum_berkedip> ')

        # Tears
        string = string.replace(":'))", ' <menangis_bahagia> ')
        string = string.replace(":')", ' <menangis_bahagia> ')
        string = string.replace(":'((", ' <menangis_sedih> ')
        string = string.replace(":'(", ' <menangis_sedih> ')
        string = string.replace("((':", ' <menangis_bahagia> ')
        string = string.replace("(':", ' <menangis_bahagia> ')

        # Some annoyed
        string = string.replace(':/', ' <terganggu> ')
        string = string.replace('(-,-)', ' <terganggu> ')
        string = string.replace('-,-', ' <terganggu> ')
        string = string.replace(':\\', ' <terganggu> ')

        # Straight face
        string = string.replace(':|', ' <muka_datar> ')
        string = string.replace(':-|', ' <muka_datar> ')

        return string
```

Kode 5.6 Fungsi Replace Emoticon

Pada potongan kode 5.6, telah ditampilkan fungsi *replace_emoticons* yang menggambarkan daftar dari emotikon beserta kata atau ekspresi yang digambarkan oleh emotikon tersebut.

```
def replace_char(self, id_data):
        print("Replace character..")
        id=0
        for line in tqdm(id_data):
            line['id'] = '0'
            line['text'] = line['text'].replace("\n", " ")
            line['text'] = re.sub(r'(https|http)?:\/\/(\w|\.|\/|\
?|\=|\&|\%)*\b', '', line['text'])
            line['text'] = re.sub(r'(?<=^|(?<=[^a-zA-Z0-9-
_\.]))#([A-Za-z]+[A-Za-z0-9-_]+)', 'hashtag', line['text'])
            line['text'] = re.sub(r'(?<=^|(?<=[^a-zA-Z0-9-
_\.]))@([A-Za-z]+[A-Za-z0-9-_]+)', 'mention', line['text'])
            line['text'] = re.sub(r"(\S)\1+", r"\1\1", line['text
'])
            line['text'] = replace_emoticons(line['text'])
```

```
            line['text'] = line['text'].replace("-", " ")
            line['text'] = line['text'].replace("..", " ")
            line['text'] = re.sub(r'[^a-zA-
Z ]', "", line['text'])
            line['text'] = line['text'].casefold()
            line['text'] = re.sub(" +", ' ', line['text'])
        df = pd.DataFrame.from_records(id_data)
        print("Saving result..")
        df.to_csv(f"final_result_new.csv", header=False, index=Fa
lse)
```

Kode 5.7 Fungsi *Replace char*

Pada potongan kode 5.7, dicantumkan fungsi *replace_char* yang digunakan untuk melakukan pembersihan data dari karakter yang tidak diperlukan, serta mengubah huruf pada teks menjadi *lowercase*. Hasil pembersihan kemudian disimpan kedalam suatu *dataframe* untuk selanjutnya diekspor menjadi suatu *file csv*.

```
def preprocess(self):
        print("Combining streaming file...")
        self.combine()
        self.into_json()

        print("Filtering language...")
        id_data = self.filter_lang()

        print("Replacing chara & saving data...")
        self.replace_char(id_data)
```

Kode 5.8 Fungsi *Preprocess*

Fungsi *preprocess* pada potongan kode 5.8 merupakan kode utama yang dijalankan ketika kelas dijalankan. Fungsi *preprocess* akan memanggil fungsi-fungsi lain secara berurutan sehingga diperoleh data yang telah di-*preprocessing*.

5.3.4 Pembuatan Databunch

Objek *databunch* pada FastAi digunakan untuk menyimpan *dataset*, dimana secara otomatis akan dilakukan *pre-processing* sesuai dengan aturan dari *library* tersebut. *Databunch* memiliki dua jenis, yaitu *TextLMDataBunch* untuk menyimpan *dataset language model*, dan *TextClasDataBunch*

untuk menyimpan dataset *classifier*. Apabila *dataset* untuk *classifier* berlabel sesuai dengan kelasnya, maka *dataset* dari *language model* hanya berlabel 0 untuk menyesuaikan dengan ketentuan dari FastAi.

Pada proses pembuatan *databunch*, akan digunakan kode yang pada kelas *PrepareTwitter*. Kelas ini akan melakukan pembagian data serta akan mengolah *dataset* menjadi *databunch*.

```
class PrepareTwitter():
    def __init__(
        self,
        data_dir='data',
        dataset='twitter',
        lm_file='lm_small.csv',
        clas_file='twitter_A.csv',
        experiment_name='subtask_A'
        ):

        self.data_dir = Path(data_dir)
        self.dataset = dataset
        self.dataset_dir = self.data_dir / self.dataset

        self.lm_file = lm_file
        self.clas_file = clas_file
        self.experiment_name = experiment_name
```

Kode 5.9 Deklarasi Variabel Kelas *PrepareTwitter*

Potongan kode 5.9 merupakan bagian dari kelas *PrepareTwitter*, dimana dilakukan pendeklarasian beberapa variabel yang dibutuhkan dalam pengolahan *databunch*. Beberapa variabel tersebut diantaranya *dataset_dir* untuk menyimpan lokasi dari data *tweet* yang akan digunakan sebagai data untuk pembuatan *language model* dan *classifier*, *lm_file* untuk menyimpan nama *language model*, dan *clas_file* untuk menyimpan nama dari *file csv* yang memiliki kumpulan data *twitter* berlabel sesuai dengan *subtask*-nya. *Experiment_name* merupakan nama dari *subtask* dari data yang akan diolah menjadi *databunch*.

```
def split_data(self):
        # for classifier
        df = pd.read_csv(self.dataset_dir/self.clas_file)
        trn_val, tst = train_test_split(df,test_size=0.2,random
_state=0)
        trn, val = train_test_split(trn_val,test_size=20/80,ran
dom_state=0)

        # for language model
        df2 = pd.read_csv(self.dataset_dir/self.lm_file)
        trn_val2, tst2 = train_test_split(df2,test_size=0.2,ran
dom_state=0)
        trn2, val2 = train_test_split(trn_val2,test_size=20/80,
random_state=0)

        # print data size
        print('Data size of classifier')
        print('Train: {}'.format(trn.shape[0]))
        print('Validate: {}'.format(val.shape[0]))
        print('Test: {}'.format(tst.shape[0]))

        print('Data size of language model')
        print('Train: {}'.format(trn2.shape[0]))
        print('Validate: {}'.format(val2.shape[0]))
        print('Test: {}'.format(tst2.shape[0]))

        # save data into files
        trn.to_csv(self.dataset_dir/'train_clas.csv',header=Non
e,index=None)
        val.to_csv(self.dataset_dir/'valid_clas.csv',header=Non
e,index=None)
        tst.to_csv(self.dataset_dir/'test_clas.csv',header=None
,index=None)

        trn2.to_csv(self.dataset_dir/'train_lm.csv',header=None
,index=None)
        val2.to_csv(self.dataset_dir/'valid_lm.csv',header=None
,index=None)
        tst2.to_csv(self.dataset_dir/'test_lm.csv',header=None,
index=None)
```

Kode 5.10 Fungsi *Split Data* Pada Pembuatan *Databunch*

Potongan kode pada kode 5.10 memperlihatkan *method split_data*. Hal pertama yang dilakukan fungsi ini adalah

membaca data *classifier* dan *language model*, kemudian melakukan pembagian antara data *train*, *test*, dan *validation data* untuk *classifier* dan *language model*. Ukuran dari *validation* dan *test data* merupakan 5% dari keseluruhan data. Maing-masing *dataset* akan disimpan kedalam objek *dataframe*. Setelah itu, ukuran dari tiap *dataset* akan dicetak. *Dataset* kemudian disimpan ke dalam beberapa *file* csv yang berbeda untuk digunakan dalam tahapan selanjutnya.

```
1.  def prepare_databunch(self):
2.      # load train, valid in df
3.      train_df = pd.read_csv(self.dataset_dir/'train_clas.csv',heade
    r=None)
4.      valid_df = pd.read_csv(self.dataset_dir/'valid_clas.csv',heade
    r=None)
5.      test_df = pd.read_csv(self.dataset_dir/'test_clas.csv',header=
    None)
6.
7.      train_df_lm = pd.read_csv(self.dataset_dir/'train_lm.csv',head
    er=None)
8.      valid_df_lm = pd.read_csv(self.dataset_dir/'valid_lm.csv',head
    er=None)
9.      test_df_lm = pd.read_csv(self.dataset_dir/'test_lm.csv',header
    =None)
10.
11.     # create databunch
12.     print("Creating databunch...")
13.
14.     exist = os.path.isfile(f'tmp/lm/itos.pkl')
15.
16.     if exist:
17.         tmp_lm = TextLMDataBunch.load(path=self.dataset_dir,cache_
    name=Path(f'tmp/lm/'))
18.     else:
19.         tmp_lm = TextLMDataBunch.from_df(path=self.dataset_dir,tra
    in_df=train_df_lm,valid_df=valid_df_lm,test_df=test_df_lm)
20.         tmp_cache_lm = Path(f'tmp/lm/')
21.         tmp_lm.save(tmp_cache_lm)
22.
23.     tmp_clas = TextClasDataBunch.from_df(path=self.dataset_dir,tra
    in_df=train_df,valid_df=valid_df,
24.         test_df=test_df,vocab=tmp_lm.vocab)
25.     tmp_cache_clas = Path(f'tmp/clas/{self.experiment_name}')
```

```
26.      tmp_clas.save(tmp_cache_clas)
```

Kode 5.11 Fungsi *Prepare Databunch*

Kode 5.11 merupakan potongan kode pada fungsi *prepare_databunch. Method* ini dijalankan untuk menyimpan *dataset* dalam bentuk *databunch,* sehingga *library* FastAi dapat memanfaatkan *dataset* untuk proses *training. Dataset* yang telah dibagi dan disimpan ke dalam format *csv* dibaca kembali sebagai sebuah *dataframe.* Langkah selanjutnya adalah dengan menyimpan data ke dalam *TextLMDataBunch* dan *TextClasDataBunch*.

Jika pada *folder* penyimpanan *TextLMDataBunch* telah terdapat *file* model, maka kode akan membaca model itu. Jika tidak, maka kode akan membuat objek *TextLMDataBunch* melalui variabel *tmp_lm.* Hal ini dikarenakan pembuatan *databunch* untuk *language model* hanya perlu dilakukan sekali dan dapat digunakan untuk *subtask* lain, tidak seperti untuk *classifier* yang harus dibuat tiap *subtask*-nya. Fungsi *from_df* akan melakukan pembuatan *databunch* dengan menggunakan file *dataframe* dari data *train*, *valid* dan *test.*

Setelah membuat *databunch* untuk *language model*, kode akan membuat *TextClasDataBunch* dengan menyimpannya pada variabel *tmp_clas.* Berbeda seperti fungsi *from_df* dari *TextLMDataBunch,* selain menggunakan parameter *train, test,* dan *valid* data, fungsi *from_df* dari objek *TextClasDataBunch* juga menggunakan parameter berupa *vocab* dari *language model.*

```
1.  def prepare(self):
2.      print('Splitting data...')
3.      self.split_data()
4.
5.      print('Preparing databunch...')
6.      self.prepare_databunch()
```

Kode 5.12 Fungsi *Prepare*

Fungsi *prepare* pada potongan kode 5.12 merupakan fungsi utama yang digunakan untuk memanggil *method* lain pada kelas *PrepareTwitter.* Mula-mula, kode akan menjalankan

fungsi *split_data* pada *dataset classfier* dan *language model*. Kemudian, kode akan menjalankan pembuatan *databunch* untuk kedua objek tersebut.

5.4 Pelaksanaan Akuisisi Data Wikipedia

Pelaksanaan akuisisi data *Wikipedia* dilaksanakan dengan menggunakan *script* pada komputer dengan sistem operasi *Linux*. *Script* ini diambil dari *github* FastAi dengan pengubahan. Secara garis besar, *script* akan menyiapkan direktori dan melakukan pengunduhan *Wikipedia dump*, serta melakukan *cloning* pada *scrip WikiExtractor* dan menjalankannya.

```
1.  ROOT="data"
2.  DUMP_DIR="${ROOT}/wiki_dumps"
3.  EXTR_DIR="${ROOT}/wiki_extr"
4.  WIKI_DIR="${ROOT}/wiki"
5.  EXTR="wikiextractor"
6.  mkdir -p "${ROOT}"
7.  mkdir -p "${DUMP_DIR}"
8.  mkdir -p "${EXTR_DIR}"
9.  mkdir -p "${WIKI_DIR}"
10.
11. LANG="id"
12. echo "Chosen language: ""${LANG}"
13. DUMP_FILE="${LANG}wiki-latest-pages-articles.xml.bz2"
14. DUMP_PATH="${DUMP_DIR}/${DUMP_FILE}"
```

Kode 5.13 Pendeklarasian Variabel pada *Script Prepare Wiki*

Kode 5.13 menunjukkan potongan kode yang digunakan untuk membuat folder-folder yang akan menyimpan data *Wikipedia*. *Root* merupakan folder utama yang akan digunakan, dimana pada kode diatas bernama 'data'. Nantinya, folder tersebut akan memiliki tiga folder didalamnya, yaitu folder *wiki_dumps* untuk menyimpan hasil unduhan dari *Wikipedia dump*, *wiki_extr* untuk menyimpan hasil ekstraksi dari *file* Wikipedia, dan *wiki* untuk menyimpan token-token yang akan dihasilkan oleh proses selanjutnya. EXTR atau *extractor* sendiri merupakan variabel yang akan digunakan pada proses mengunduhan kode untuk ekstraksi data *Wikipedia*. Selain itu, terdapat pula variabel untuk menyimpan bahasa dari *Wikipedia dump* yang dibutuhkan pada LANG, nama *file* yang

akan diunduh pada DUMP_FILE, dan tempat untuk menyimpan hasil unduhan pada DUMP_PATH.

```
if [ ! -f "${DUMP_PATH}" ]; then
  read -r -p "Continue to download (WARNING: This might be big and can take a long time!) (y/n)? " choice
  case "$choice" in
    y|Y ) echo "Starting download...";;
    n|N ) echo "Exiting";exit 1;;
    * ) echo "Invalid answer";exit 1;;
  esac
  wget -c "https://dumps.wikimedia.org/""${LANG}""wiki/latest/""${DUMP_FILE}""" -P "${DUMP_DIR}"
else
  echo "${DUMP_PATH} already exists. Skipping download."
fi

# Check if directory exists
if [ ! -d "${EXTR}" ]; then
  git clone https://github.com/attardi/wikiextractor.git
  cd "${EXTR}"
  python setup.py install
fi
```

Kode 5.14 Pengunduhan Data Wikipedia dan Installasi WikiEtractor

Potongan kode 5.14 menunjukkan proses dari pengunduhan *Wikipedia dump*. Sebelum melakukan pengunduhan, terlebih dahulu dilakukan pengecekan apakah *dump file* telah ada pada direktori yang ditentukan. Jika *file* sudah ada, maka proses pengunduhan dilewati. Jika belum, maka proses pengunduhan dilanjutkan dengan memberikan masukan "y" atau "Y" pada pertanyaan. Jika jawaban dari pertanyaan itu "n", "N", atau masukan lain yang tidak valid, maka sistem akan keluar dari proses pengunduhan. Jika jawabannya "y" atau "Y", maka proses akan dilanjutkan dengan pengunduhan *file* pada *url* https://dumps.wikimedia.org/. Kode 5.14 juga menunjukkan proses *clone* pada kode *WikiExtractor* yang digunakan untuk melakukan ekstraksi pada data *wikipedia*.

```
1.  EXTR_PATH="${EXTR_DIR}/${LANG}"
2.  if [ ! -d "${EXTR_PATH}" ]; then
3.    read -r -
    p "Continue to extract Wikipedia (WARNING: This might take a lo
    ng time!) (y/n)? " choice
4.    case "$choice" in
5.      y|Y ) echo "Extracting ${DUMP_PATH} to ${EXTR_PATH}...";;
6.      n|N ) echo "Exiting";exit 1;;
7.      * ) echo "Invalid answer";exit 1;;
8.    esac
9.    python wikiextractor/WikiExtractor.py -s --json -
    o "${EXTR_PATH}" "${DUMP_PATH}"
10. else
11.   echo "${EXTR_PATH} already exists. Skipping extraction."
12. fi
```

Kode 5.15 Proses Ekstraksi Data Wikipedia

Kode 5.15 melaksanakan proses installasi *Wikiextractor* yang nantinya akan digunakan untuk melakukan ekstraksi pada *file Wikipedia dump*. Setelah dilakukan installasi, kode akan mencari tau apakah folder ekstraksi *file Wikipedia* telah ada sebelumnya. Jika sudah, kode akan tidak melakukan proses ekstraksi. Sebaliknya, apabila proses ekstraksi belum dilaksanakan, maka kode akan melakukan konfirmasi apakah pengguna hendak mengekstraksi data dari *Wikipedia*. Apabila iya, maka kode baru akan menjalankan proses ekstraksi dengan *wikiextractor* pada *file Wikipedia dump* yang memiliki format XML. Hasil dari proses ini adalah beberapa dokumen dengan format json digunakan untuk menyimpan tiap artikel.

5.5 Pelaksanaan Pre-processing Data Wikipeda

Masukan dari proses ini adalah beberapa *file* dengan format json yang menyimpan data-data artikel dari *Wikipedia* bahasa Indonesia. Sebelum dilaksanakan pengolahan lebih lanjut untuk pembuatan model bahasa, dilakukan beberapa langkah *pre-preprocessing* agar data dapat dimanfaatkan dengan optimal.

5.5.1 Pembagian Dataset *Wikipedia*

Pada langkah ini, dilakukan pembagian *dataset* menjadi data untuk *training*, *testing*, dan *validation*. Berdasarkan petunjuk dari FastAi, ukuran dari data *training* akan mengikuti

ukuran maksimal jumlah token yang dapat diperoleh dalam dari data *Wikipedia*. Sedangkan data untuk *testing* dan *validation* pada tugas akhir ini dibuat sekitar 200.000 token. Pada penjelasan di sub bab ini, akan dijelaskan kode *create_wikitext* untuk membagi dataset *Wikipedia*.

```
1.  def get_texts(root):
2.      for dir_ in root.iterdir():
3.          for wiki_file in dir_.iterdir():
4.              with open(wiki_file, encoding='utf-8') as f_in:
5.                  for line in f_in:
6.                      article = json.loads(line)
7.                      text = article['text']
8.                      title = article['title']
9.                      if text.strip() == title:
10.                         continue
11.                     yield text
```

Kode 5.16 Fungsi get_texts pada Create_wikitext

Potongan kode 5.16 merupakan *method get_texts* yang digunakan untuk mengumpulkan teks artikel dari *Wikipedia dump* yang memiliki format *json* menjadi berformat teks. Parameter yang digunakan oleh fungsi ini adalah *root* berupa folder dimana kumpulan *Wikipedipedia dump* berada. Fungsi *get_text* akan melakukan iterasi dalam folder *root* untuk memperoleh seluruh *file Wikipedia dump* dari dalam masing-masing sub folder, kemudian dari tiap *file* akan diperoleh teks artikel yang akan digunakan dalam proses berikutnya.

```
1.  def write_wikitext(file_path, text_iter, mt, num_tokens, mode='w')
    :
2.      total_num_tokens = 0
3.      print(f'Writing to {file_path}...')
4.      i = 0
5.      with open(file_path, mode, encoding='utf-8') as f_out:
6.          for i, text in enumerate(text_iter):
7.
8.              num_tokens_article = 0
9.
10.             tokenized_paragraphs = []
11.             paragraphs = text.split('\n')
```

```
12.
13.             for paragraph in paragraphs:
14.                 tokenized = mt.tokenize(paragraph.strip(), return_
    str=True)
15.                 tokenized_paragraphs.append(tokenized)
16.
17.                 tokens = tokenized.split(' ')
18.                 tokens = [token for token in tokens if token]
19.
20.
21.                 num_tokens_article += len(tokens) + 1
22.
23.             if num_tokens_article < 100:
24.                 continue
25.
26.             for tokenized in tokenized_paragraphs:
27.                 f_out.write(tokenized + '\n')
28.
29.             total_num_tokens += num_tokens_article + 1
30.             if num_tokens is not None and total_num_tokens > num_t
    okens:
31.                 break
32.             if i % 10000 == 0 and i > 0:
33.                 print('Processed {:,} documents. Total # tokens: {
    :,}.'.format(i, total_num_tokens))
34.     print('{}. # documents: {:,}. # tokens: {:,}.'.format(
35.         file_path, i, total_num_tokens))
```

Kode 5.17 Fungsi Write_wikitext

Setelah teks dari *file Wikipedia dump* diperoleh, maka langkah selanjutnya adalah melakukan tokenisasi. Dalam proses yang ditunjukkan pada potongan kode 5,17, *method write_wikitext* memiliki beberapa parameter. Parameter *file_path* digunakan untuk memasukkan direktori dimana *file* akan disimpan beserta nama dari file, *text_iter* akan menyimpan kumpulan teks yang diambil dari *Wikipedia dump* sebagai suatu variabel, *mt* merupakan *tokenizer* yang digunakan, *num_tokens* mendeklarasikan jumlah token yang harus diambil dari teks.

Pertama-tama, kode akan mengambil tiap artikel yang dilambangkan dengan variabel *i*. Kode akan memisahkan tiap paragraf, kemudian dilakukan tokenisasi pada tiap paragraf. Kemudian proses tokenisasi baru dilanjutkan pada tiap-tiap kata

dan disimpan dalam variabel *tokens*. *Num_tokens_article* akan menyimpan total token yang telah diperoleh. Jika artikel tersebut memiliki *num_tokens_article* atau total token kurang dari seratus, maka token dari artikel tersebut tidak digunakan, dan proses tokenisasi dilanjutkan pada artikel lain. Jika jumlah token yang dibutuhkan telah memenuhi ketentuan, maka proses tokenisasi dihentikan. Selanjutnya, akan dilakukan penyimpanan jumlah dokumen (artikel) dan token yang telah diproses kedalam suatu *file* dengan *format csv*.

```
1.  def main(args):
2.
3.      input_path = Path(args.input)
4.      output = Path(args.output)
5.      assert input_path.exists(), f'Error: {input_path} does not ex
    ist.'
6.      output.mkdir(exist_ok=True)
7.
8.      mt = MosesTokenizer(args.lang)
9.
10.     lrg_wiki = output / f'{args.lang}-100'
11.     lrg_wiki.mkdir(exist_ok=True)
12.
13.     text_iter = get_texts(input_path)
14.
15.     splits = ['valid', 'test', 'train']
16.     token_nums = [200000, 200000, 100000000]
17.     for split, token_num in zip(splits, token_nums):
18.         lrg_file_path = lrg_wiki / f'{args.lang}.wiki.{split}.tok
    ens'
19.         write_wikitext(lrg_wiki_train, text_iter, mt, token_num)
```

Kode 5.18 Method Main pada Create_Wikitext

Pada kode 5.18, digambarkan bagaimana proses keseluruhan pembuatan *wikitext* berlangsung. Direktori masukan dan luaran dari *file* akan disimpan kedalam variabel *input_path* dan *output*. Variabel *mt* adalah suatu *tokenizer* berupa *MosesTokenizer* untuk bahasa Indonesia. Luaran dari *wikitext* akan disimpan pada direktori *lrg_wiki*. Variabel

text_iter sendiri akan melakukan pengambilan data teks menggunakan fungsi *get_texts*.

Splits menyimpan tiga jenis *dataset* yang akan dihasilkan dalam pembuatan *wikitext*, yaitu dataset untuk *train*, *valid*, dan *test.* Sedangkan *token_num* menyimpan jumlah token minimal yang akan digunakan pada masing-masing *dataset*. Untuk data *train*, akan digunakan token sejumlah kurang lebih 100 juta token. Namun jika kurang dari itu, maka token yang akan digunakan adalah seluruh token pada data *Wikipedia*. Sedangkan untuk data *valid* dan data *test*, digunakan token sejumlah lebih dari 200.000 token.

Proses selanjutnya adalah dilakukan suatu iterasi dimana akan dipanggil fungsi *write_wikitext* untuk *membuat dataset* masing-masing *split (train*, *valid*, dan *test)* dengan ukuran sesuai *dengan token_nums*. Proses ini akan menghasilkan sebuah folder, yaitu folder id-100, dimana pada folder tersebut, akan tersimpan tiga buah *dataset* untuk proses *train*, *test*, dan *validation.*

5.5.2 *Pre-processing Dataset Wikipedia*

Pre-processing dataset Wikipedia dilakukan setelah dataset telah terbagi. Beberapa tahapan yang dilalui diantaranya pembuatan *vocabulary*, pembatasan ukuran *vocab*, dan mengganti

```
def build_vocab(file_path, cutoff=3):
    counter = Counter()
    with open(file_path, 'r', encoding='utf-8') as f:
        for i, line in enumerate(f):
            tokens = line.strip().split(' ') + ['<eos>']
            counter.update(tokens)
    vocab = {}
    in_vocab_count = 0
    OOV_count = 0
    for token, count in counter.most_common():
        if count >= cutoff:
            vocab[token] = count
            in_vocab_count += count
        else:
            OOV_count += count
```

```
16.     print('OOV ratio: %.4f.' % (OOV_count / (in_vocab_count + O
    OV_count)))
17.     return vocab
```

Kode 5.19 Fungsi Build Vocab pada Postprocess Wikitext

Potongan kode 5.19 menujukkan bagaimana kode melakukan pembuatan *vocabulary* pada data *training* dari *Wikipedia*. Dengan menggunakan *counter*, fungsi akan menghitung frekuensi dari munculnya suatu token. Selain itu, akan dihitung nilai rasio OOV atau rasio kata *out-of-vocabulary*. OOV merupakan kata-kata yang muncul pada *testing data* namun tidak terdapat pada *vocabulary*.

```
1.  def limit_vocab(unk_path, vocab):
2.      temp_file_path = unk_path.with_name(unk_path.name + '.temp'
    )
3.      total_num_tokens = 0
4.      print(f'Limiting vocab in {unk_path}. Writing to {unk_path}
    .')
5.      with open(unk_path, 'r', encoding='utf-
    8') as f_in, open(temp_file_path, 'w', encoding='utf-
    8') as f_out:
6.          for line in f_in:
7.              tokens = [x for x in line.strip().split(' ') if x]

8.              tokens = [token if token in vocab else UNK for toke
    n in tokens]
9.              # Ensures there's a space between tokens, including
     the last word,
10.             # newline, and the first word of the next line
11.             tokens = tokens + ['\n']
12.             total_num_tokens += len(tokens)
13.             tokens = [''] + tokens
14.             line = ' '.join(tokens)
15.             f_out.write(line)
16.     print(f'{unk_path.name}. #
    of tokens: {total_num_tokens}')
17.     temp_file_path.replace(unk_path)
```

Kode 5.20 Fungsi Limit Vocab pada Postprocess_wiki

Pada *method limit_vocab* yang terdapat pada kode 5.20, akan dilakukan pembatasan terhadap kata yang ada pada data

testing dan *validation*. Jika pada data tersebut terdapat kata yang tidak tercantum pada *vocabulary*, maka kata tersebut akan diganti dengan variabel UNK *(unknown).*

```
def replace_numbers(file_path, unk_path):
    print(f'Replacing numbers in {file_path}. Writing to {unk_path}.')
    with open(file_path, 'r', encoding='utf-8') as f_in, open(unk_path, 'w', encoding='utf-8') as f_out:
        for line in f_in:
            raw_tokens = line.strip().split(' ')
            tokens = []
            for token in raw_tokens:
                tokens.append(replace_number(token))
            tokens = [''] + tokens + ['\n']
            line = ' '.join(tokens)
            f_out.write(line)
```

Kode 5.21 Fungsi Replace_numbers pada Postprocess_wiki

Pada *method replace_numbers* yang ditunjukkan pada potongan kode 5.21, setiap token akan mengalami proses *replace_number* menggunakan kode yang telah disediakan oleh *FastAi.* Proses ini akan melakukan penggantian format angka yang mengandung titik, dengan memisahkan titik dengan angka yang ada. Jika masukan dari fungsi *replace_number* adalah angka *"1.23"*, maka luaran dari fungsi tersebut adalah *"1 @.@ 23"*.

5.6 Pembuatan Model ULMFiT

5.6.1 Pembuatan *Pretrained Model*

Proses pembuatan *pretrained model* merupakan tahapan pertama dalam pembuatan model pada ULMFiT. Pada tahapan ini, korpus yang berasal dari *Wikipedia* dan data *Twitter* tanpa label akan digunakan sebagai data masukan. *Library* yang digunakan dalam pembuatan *pretrained model* ini adalah *FastAi.*

```
def pretrain_lm(dir_path, lang='id', cuda_id=0, bs=32, bptt=70, name='wt-103', num_epochs=10, max_vocab=60000):

    results = {}
    model_dir = 'models'
    if not torch.cuda.is_available():
```

```
            print('CUDA not available. Setting device=-1.')
            cuda_id = -1
        torch.cuda.set_device(cuda_id)
```

Kode 5.22 Fungsi Pretrain Model

Kode untuk pembuatan *pretrained model* diawali dengan pembuatan fungsi *pretrain_lm* dimana beberapa parameter *default* ditentukan. Pada potongan kode 5.22, terdapat pula perintah untuk menggunakan GPU apabila tersedia.

```
dir_path = Path(dir_path)
assert dir_path.exists()
model_dir_itos = Path(dir_path / model_dir)
model_dir_itos.mkdir(exist_ok=True)
print('Batch size:', bs)
model_name = 'lstm'

trn_path = dir_path / f'{lang}.wiki.train.tokens'
val_path = dir_path / f'{lang}.wiki.valid.tokens'
tst_path = dir_path / f'{lang}.wiki.test.tokens'
for path_ in [trn_path, val_path, tst_path]:
    assert path_.exists(), f'Error: {path_} does not exist.'
```

Kode 5.23 Pendeklarasian Variabel Pretrain LM

Pada potongan kode 5.23, dilakukan pendeklarasian variabel direktori dari tiap *file* yang akan digunakan untuk proses *pretraining*, serta pengecekan apakah direktori tersebut tersedia.

```
trn_tok = read_whitespace_file(trn_path)
val_tok = read_whitespace_file(val_path)
```

Kode 5.24 Pengambilan Token Train dan Val

Pada potongan kode 5.24, dilakukan pembacaan file dari *trn_path* dan *val_path*. *Read_whitespace_file* merupakan suatu fungsi yang berasal dari *fastai_contrib.utils*, dimana fungsi tersebut digunakan untuk mengambil token yang dipisahkan dengan spasi pada suatu data.

```
# create the vocabulary
cnt = Counter(word for sent in trn_tok for word in sent)
```

```
itos = [o for o,c in cnt.most_common(n=max_vocab)]
itos.insert(1, PAD)
assert UNK in itos, f'Unknown words are expected to have been r
eplaced with {UNK} in the data.'

vocab = Vocab(itos)
stoi = vocab.stoi
```

Kode 5.25 Pembuatan Vocabulary pada Pretrain LM

Potongan kode 5.25, digunakan suatu *counter* untuk melakukan penghitungan jumlah kata pada korpus, kemudian data dari *vocabulary* akan disimpan dalam *itos*. PAD yang digunakan untuk membuat setiap kalimat memiliki panjang yang sama akan disimpan dengan id 1. Kemudian akan dilakukan pengecekan apakah korpus memiliki token UNK, jika ada, maka proses dapat dilanjutkan.

```
# save vocabulary
itos_fname = model_dir_itos / f'itos_{name}.pkl'
print(f"Saving vocabulary as {itos_fname}")
results['itos_fname'] = itos_fname
with open(itos_fname, 'wb') as f:
    pickle.dump(itos, f)
```

Kode 5.26 Penyimpanan Vocabulary Pretrain LM

Pada kode 5.26, dilakukan penyimpanan *vocabulary* menjadi suatu file *itos* dengan ekstensi *.pkl.*

```
trn_ids = np.array([([stoi.get(w, stoi[UNK]) for w in s]) for s
 in trn_tok])
val_ids = np.array([([stoi.get(w, stoi[UNK]) for w in s]) for s
 in val_tok])

data_lm = TextLMDataBunch.from_ids(path=dir_path, vocab=vocab,
train_ids=trn_ids,valid_ids=val_ids, bs=bs, bptt=bptt)
```

Kode 5.27 Pembuatan TextLMDataBunch Pretrain LM

Langkah selanjutnya ditulis pada kode 5.27, dimana dilakukan pembuatan *data_lm* dengan menggunakan *TextLMDataBunch* dengan *menggunakan* id dari data *trn_tok* dan *val_tok. TextLMDataBunch* merupakan suatu fungsi dalam

library fastai dimana *dataset* disimpan serta dilakukan *pre-processing* secata otomatis. *TextLMDataBunch* sendiri dibuat untuk *meyimpana dataset* untuk proses *training* pada *language model.*

```
emb_sz, nh, nl = 400, 1150, 3
dps = np.array([0.25, 0.1, 0.2, 0.02, 0.15])
drop_mult = 0.1

fastai.text.learner.default_dropout['language'] = dps * drop_mu
lt
learn = language_model_learner(data_lm, bptt=bptt, emb_sz=emb_s
z, nh=nh, nl=nl, pad_token=1,drop_mult=drop_mult, model_dir=mod
el_dir,clip=0.12, callback_fns=[partial(CSVLogger, filename="hi
story_model_0")])
try:
    os.remove('history_model_0.csv')
except OSError:
    pass
```

Kode 5.28 Pembuatan Objek Language Model Learner

Pada potongan kode 5.28, diperlihatkan proses dimana dimasukkan *hyperparameter* yang akan digunakan dalam *pretrained model*. Setelah *hyperparameter* ditentukan, proses selanjutnya adalah peyimpanan fungsi *language_model_learner* pada variabel *learn. Language_model_learner* akan menghasilkan objek *Learn* dimana merupakan suatu *trainer* pada data untuk meminimalisir *loss* dengan menggunakan suatu fungsi optimasi. Tujuan dibuatnya objek *Learner* adalah agar dapat dilakukan *train* pada model menggunakan fungsi *Learner.fit*, dimana pada setiap *epoch*, semua *metrics* akan dicetak. Pada *callback* dari objek *learner*, digunakan *callback CSVLogger.* Fungsi dari CSVLogger adalah untuk menyimpan hasil *train_loss, val_loss* serta akurasi dari tiap epoch saat *training.*

```
lr = 5e-3
wd = 1e-7
div_factor = None
pct_start = None
learn.opt_fn = partial(optim.Adam, betas=(0.8, 0.99))
```

```
learn.true_wd = False

fit_one_cycle(learn, num_epochs, lr, moms=(0.8, 0.7), wd=wd,cal
lbacks=[SaveModelCallback(learn, every='improvement', monitor='
accuracy', name='best')])
```

Kode 5.29 Proses Training Pretrain LM

Potongan kode 5.29 menunjukkan pemanggilan fungsi *fit_one_cycle* untuk dilakukan proses pelatihan pada objek *learn*. *Fit_one_cycle* digunakan untuk pelatihan pada model menggunakan aturan 1cycle. *Callback* yang digunakan adalah *SaveModelCallback* yang berfungsi untuk menyimpan model dengan akurasi terbaik dengan nama *file 'best'*.

```
print(f"Saving models at {learn.path /learn.model_dir}")
learn.save(f'{model_name}_{name}')

opt_state_path = learn.path / learn.model_dir / f'{model_name}3
_{name}_state.pth'
print(f"Saving optimiser state at {opt_state_path}")
torch.save(learn.opt.opt.state_dict(), opt_state_path)

results['accuracy'] = learn.validate()[1]
return results
```

Kode 5.30 Penyimpanan Model Pretrain LM

Pada kode 5.30, akan dilakukan penyimpanan model serta *optimiser state* dari model. Setelah semua proses dilakukan, maka fungsi akan mengembalikan nilai akurasi serta nama model yang telah disimpan.

5.6.2 Pelaksanaan *Fine-tuning Language Model*

Proses *finetuning* pada *pretrained model* dimaksudkan agar model bahasa dapat menyesuaikan dengan ciri kebahasaan dari task classifier yang akan dilaksanakan. Pada sub bab ini, akan dibahas mengenai fungsi-fungsi yang digunakan dalam proses *fine-tuning* pada *language model.*

```
class Finetuning_Twitter():

    def __init__(
        self,
```

```
        data_dir='data',
        lang='id' ,
        cuda_id=0 ,
        pretrain_name='wt-103',
        model_dir='data/wiki/id/id-100-unk/models',
        dataset='twitter',
        bs=64,
        bptt=70,
        pad_token=PAD_TOKEN_ID,
        fine_tune_lm=True,
        lm_lr=1e-2,
        lm_drop_mult=0.5,
        lm_epochs=3,
        deterministic = True,
        experiment_name = 'full',
        ):

        # set parameters
        self.data_dir = Path(data_dir)
        self.dataset = dataset
        self.dataset_dir = self.data_dir / self.dataset
        self.pretrain_name = pretrain_name
        self.model_dir = Path(model_dir)
        self.bs = bs
        self.bptt = bptt
        self.pad_token = pad_token
        self.lm_epochs = lm_epochs
        self.fine_tune_lm = fine_tune_lm
        self.lm_lr = lm_lr
        self.lm_drop_mult = lm_drop_mult
        self.deterministic = deterministic
        self.experiment_name = experiment_name
```

Kode 5.31 Pendeklarasian Variabel Finetuning

Pada potongan kode 5.31, dilakukan proses pendeklarasian variabel yang akan digunakan pada proses *finetuning* pada *pretrained model*.

```
# set network parameters
self.emb_sz, self.nh, self.nl = 400, 1150, 3
# use gpu if available
if not torch.cuda.is_available():
    print('CUDA not available. Setting device=-1.')
    cuda_id = -1
```

```
7.   torch.cuda.set_device(cuda_id)
8.   # enable/disable cuda if needed. param: cuda & cpu
9.   defaults.device = torch.device('cuda')
```

Kode 5.32 Pendeklarasian Network Parameter dan Penggunaan GPU

Pada potongan kode 32, dilakukan pendeklarasian parameter untuk pembuatan model, serta dilakukan pula pengecekan apakah GPU tersedia pada perangkat. Secara *default*, *torch* akan menggunakan GPU dengan *id* 0. Apabila tidak ada, maka *id device* yang digunakan adalah -1. Pengaturan *defaults.device* pada *FastAi* menggunakan CPU dilakukan apabila percobaan membutuhkan pembanding dengan menggunakan *device* CPU.

```
1.  # set names
2.  self.model_name = 'lstm'
3.
4.  self.lm_name = f'best'
5.  self.pretrained_fnames = (self.lm_name, f'itos_{self.pretrain_name}
    ')
6.  self.lm_enc_finetuned  = f"{self.lm_name}_{self.dataset}_{self.expe
    riment_name}_enc"
7.
8.  # ensure path exists
9.  ensure_paths_exists(self.data_dir,
10.             self.dataset_dir,
11.             self.model_dir,
12.             self.model_dir/f"{self.pretrained_fnames[0]}.pth",
13.             self.model_dir/f"{self.pretrained_fnames[1]}.pkl")
```

Kode 5.33 Pendeklarasian Variabel Nama dan Lokasi Model Finetuning

Potongan kode 5.33 menunjukkan proses pendefinisan serta pengecekan apakah *pretrained model* telah tersedia pada direktori yang telah didefinisikan.

```
1.  def set_deterministic(self):
2.      if self.deterministic:
3.          torch.backends.cudnn.deterministic = True
4.          torch.backends.cudnn.benchmark = False
5.          torch.manual_seed(0)
6.          np.random.seed(0)
```

```
7.      print(f'Set deterministic done')
```

Kode 5.34 Fungsi Set Deterministik Finetuning

Pada potongan kode 5.34, ditunjukkan sebuah fungsi *set_deterministic*, dimana fungsi ini digunakan untuk melakukan pengaturan ulang pada *seed torch* serta pada *seed* dari *numpy* agar dapat hasil dari suatu proses yang membutuhkan *torch* ataupun *numpy* dapat di reproduksi kembali.

```
1.  def fit_triangular(self, learn, epoch, lr, clas_experimen
    t):
2.      div = 20
3.      split = 5
4.      n = len(learn.data.train_dl)
5.      print(n)
6.      phases=[]
7.      phases.append(TrainingPhase(lrs=(lr/div, lr), length=
    n*(epoch*(1/split)), moms=(0.99, 0.8)))
8.      phases.append(TrainingPhase(lrs=(lr, lr/div), length=
    n*(epoch*((split-1)/split)), moms=(0.8,0.99)))
9.      for i in range(0,2):
10.         phases[i].lr_anneal=annealing_cos
11.         phases[i].mom_anneal = annealing_cos
12.     sched = GeneralScheduler(learn, phases)
13.     learn.callbacks.append(sched)
14.     learn.fit(epoch,
15.         callbacks=[SaveModelCallback(learn, every='improv
    ement', monitor='accuracy', name=f'best_{self.experiment_
    name}_{clas_experiment}')])
```

Kode 5.35 Fungsi Fit Triangular Finetuning

Pada kode 5.35, telah dicantumkan fungsi *fit_triangular* yang digunakan untuk proses *training* dengan aturan *slanted triangular learning rates.*

```
1.  def lm_fine_tuning(self, experiment_name, data_lm):
2.
3.      learn = language_model_learner(
4.          data_lm,
5.          bptt=self.bptt,
```

```
        emb_sz=self.emb_sz,
        nh=self.nh,
        nl=self.nl,
        pad_token=self.pad_token,
        pretrained_fnames=self.pretrained_fnames,
        path=self.model_dir.parent,
        model_dir=self.model_dir.name,
        drop_mult=self.lm_drop_mult,
        callback_fns=[partial(CSVLogger, filename=f'history_lm_{self.experiment_name}')])

    try:
        os.remove(f'history_lm_{self.experiment_name}.csv')
    except OSError:
        pass

    self.set_deterministic()

    if experiment_name=='full':
        print('Fine-tuning the language model with full scenario...')
        learn.unfreeze()
        learn.fit(self.lm_epochs, self.lm_lr, callbacks=[SaveModelCallback(learn, every='epoch', monitor='accuracy', name=f'best_{experiment_name}')])
    elif experiment_name=='full_discr':
        print('Fine-tuning the language model with full and discr scenario...')
        learn.unfreeze()
        lrs = np.array([self.lm_lr/6,self.lm_lr/3,self.lm_lr,self.lm_lr/2])
        learn.fit(self.lm_epochs, lrs,  callbacks=[SaveModelCallback(learn, every='epoch', monitor='accuracy', name=f'best_{experiment_name}')])
    elif experiment_name=='full_discr_sltr':
        print('Fine-tuning the language model with full, discr and sltr scenario...')

        lrs = np.array([self.lm_lr/6,self.lm_lr/3,self.lm_lr,self.lm_lr/2])
        self.fit_triangular(learn, self.lm_epochs, lrs, 3, experiment_name)
    elif experiment_name=='full_sltr':
        print('Fine-tuning the language model with full and sltr scenario...')
```

```
38.         self.fit_triangular(learn, self.lm_epochs, self.lm_lr, 3,
    experiment_name)
39.     else:
40.         print('Experiment name doesn\'t valid')
41.
42.     print(f"Saving models at {learn.path / learn.model_dir}")
43.     learn.save_encoder(f"{self.lm_enc_finetuned}")
44.
45.     return learn
```

Kode 5.36 Fungsi lm_fine_tuning

Pada kode 5.36, ditunjukkan fungsi *lm_fine_tuning* yang digunakan dalam proses utama *fine-tuning*. Pertama, fungsi akan melakukan pembuatan objek *language_model_learner* menggunakan *data_lm* yang merupakan *databunch* dari *dataset tweet* dan *pretrained model* menggunakan *Wikipedia*. Kemudian, fungsi akan melakukan *fine-tuning* pada *language_model_learner* menggunakan skenario yang dipilih. Ada empat skenario yang dapat digunakan, yaitu *full*, *full_discr*, *full_discr_sltr*, dan *full_sltr*. Setelah mengalami proses *fine-tuning*, model akan disimpan kedalam suatu *encoder*.

```
1.  def fine_tune(self):
2.      data_lm = TextLMDataBunch.load(path=Path(self.data_dir/self.da
    taset),cache_name=Path(f'tmp/lm'))
3.  self.lm_fine_tuning(self.experiment_name, data_lm)
```

Kode 5.37 Fungsi fine_tune

Fungsi *fine_tune* pada kode 5.37 digunakan untuk melakukan pemanggilan pada *TextLMDataBunch* untuk kemudian digunakan sebagai objek *finetuning.*

5.6.3 Pembuatan *Classifier*

Proses pembuatan *classifier* dilakukan dengan memanfaatkan objek *text_classifier_learner* yang merupakan *learner* yang menyimpan *databunch* untuk proses *training*. Proses ini akan dilangsungkan pada suatu kelas bernama *Classifying_Twitter*.

```
1.  class Classifying_Twitter():
2.
```

```
    def __init__(
        self,
        data_dir='data',
        lang='id' ,
        cuda_id=0 ,
        pretrain_name='wt-103',
        model_dir='data/wiki/id/id-100-unk/models',
        fig_dir='data/wiki/id/id-100-unk/figs',
        dataset='twitter',
        bs=64,
        bptt=70,
        pad_token=PAD_TOKEN_ID,
       use_pretrained_lm = True,
        fine_tune_lm=True,
        clas_lr=1e-2,
        clas_drop_mult=0.1,
        clas_epoch=15,
        use_lm=True,
        clas_lr_discount=2.6**4,
        deterministic = True,
        experiment_name = 'subtask_C',
        lm_scenario = "full_discr"
        ):

        # set parameters
        self.data_dir = Path(data_dir)
        self.dataset = dataset
        self.dataset_dir = self.data_dir / self.dataset
        self.pretrain_name = pretrain_name
        self.model_dir = Path(model_dir)
        self.fig_dir = Path(fig_dir)
        self.fig_dir.mkdir(exist_ok=True)
        self.bs = bs
        self.bptt = bptt
        self.pad_token = pad_token
        self.fine_tune_lm = fine_tune_lm
        self.use_pretrained_lm = use_pretrained_lm
        self.clas_lr = clas_lr
        self.clas_drop_mult = clas_drop_mult
        self.clas_epoch = clas_epoch
        self.use_lm = use_lm
        self.deterministic = deterministic
        self.experiment_name = experiment_name
        self.lm_scenario = lm_scenario

```

```
48.         # set network parameters
49.         self.emb_sz, self.nh, self.nl = 400, 1150, 3
50.
51.         # use gpu if available
52.         if not torch.cuda.is_available():
53.             print('CUDA not available. Setting device=-1.')
54.             cuda_id = -1
55.         torch.cuda.set_device(cuda_id)
56.
57.         # enable/disable cuda if needed. param: cuda & cpu
58.         defaults.device = torch.device('cuda')
59.
60.         self.model_name = 'lstm'
61.
62.         self.lm_name = f'best'
63.         self.pretrained_fnames = (self.lm_name, f'itos_{self.pretra
    in_name}')
64.         self.lm_enc_finetuned  = f"{self.lm_name}_{self.dataset}_{s
    elf.lm_scenario}_enc"
65.         #self.cm_name = self.fig_dir / f'{self.experiment_name}_{se
    lf.lm_scenario}_confusionmatrix.png'
66.
67.         # ensure path exists
68.         ensure_paths_exists(self.data_dir,
69.                             self.dataset_dir,
70.                             self.model_dir,
71.                             self.model_dir/f"{self.pretrained_fname
    s[0]}.pth",
72.                             self.model_dir/f"{self.pretrained_fname
    s[1]}.pkl")
73.
74.         # show dataset and language
75.         print(f'Dataset: {dataset}. Language: {lang}.')
```

Kode 5.38 Pendeklarasian Variabel Classifier

Pada potongan kode 5.38, dilakukan pendeklarasian variabel dan parameter untuk kelas *Classifying_Twitter*. Selain itu, juga dilakukan pengaturan GPU yang digunakan oleh kode.

```
1.   def classify_full(self, learn):
2.
3.       class_scenario = 'full'
4.
5.       print("Classify twitter with full scenario")
```

```
6.
7.      learn.callback_fns.append(partial(CSVLogger, filename=f'histor
    y_clas_{self.experiment_name}_{class_scenario}'))
8.
9.      learn.model.reset()
10.     if self.use_lm:
11.         print('Loading language model')
12.         lm_enc = self.lm_enc_finetuned
13.         learn.load_encoder(lm_enc)
14.     else:
15.         print('Training from scratch without language model')
16.
17.     # train
18.     self.set_deterministic()
19.
20.     learn.unfreeze()
21.     learn.fit(self.clas_epoch, self.clas_lr, wd=1e-7,
22.         callbacks=[SaveModelCallback(learn, every='improvement', m
    onitor='accuracy', name=f'best_classifier_{self.experiment_name}_{
    class_scenario}')])
23.
24.     # getting accuracy results
25.     results = {}
26.     results['accuracy'] = learn.validate()[1]
27.     print(results)
28.
29.     # save classifier
30.     print(f"Saving models at {learn.path / learn.model_dir}")
31.     learn.save(f'{self.model_name}_{self.experiment_name}_{class_s
    cenario}_class')
32.
33.     # getting prediction on validation data
34.     preds = learn.get_preds()
35.
36.     # getting prediction only
37.     p = preds[0]
38.     p = p.tolist()
39.     p = np.asarray(p)
40.
41.     # binarize
42.     y_pred = p.argmax(axis=1)
43.
44.     # target
45.     t = preds[1].tolist()
46.     y_true = np.asarray(t)
```

```
47.
48.     return y_true, y_pred
```

Kode 5.39 Fungsi Classify Full pada Classifier

Pada potongan kode 5.39, terdapat fungsi *classify_full* yang merupakan salah satu skenario pembuatan *classifier* dengan training pada seluruh layer. Masukan dari fungsi tersebut adalah objek *learner* yang dibuat dengan memanggil *text_classifier_learner* dan menggunakan *databunch* dari data *subtask* yang dibutuhkan. *Encoder* dari *language model* yang telah mengalami proses *fine-tuning* akan dimuat pada objek *learner*. Selanjutnya, proses *fit* pada *classifier* akan diljalankan berdasarkan skenario yang dibutuhkan. *Classifier* yang telah di *training* akan disimpan, kemudian akan diperoleh hasil prediksi dari objek *learner* tersebut.

5.7 Evaluasi Pengklasifikasi

Proses evaluasi dari pengklasifikasi dilakukan dengan memanfaatkan masukkan berupa *y_pred* dan *y_true* yang merupakan luaran dari proses sebelumnya. Proses ini dilakukan pada fungsi *evaluation*, lalu dilanjutkan dengan fungsi *plot_confusion_matrix* untuk membuat *confusion matrix* dan *safe_confusion_matrix* untuk menyimpan hasil dari *confusion matrix* yang telah dihasilkan.

```
1.  def evaluation(self,y_true,y_pred, class_scenario):
2.      "Evaluate the classification results"
3.
4.      ## metrics
5.      f1_micro = f1_score(y_true,y_pred,average='micro')
6.      f1_macro = f1_score(y_true,y_pred,average='macro')
7.      f1_weighted = f1_score(y_true,y_pred,average='weighted')
8.
9.      print('F1 score')
10.     print('micro: {}'.format(f1_micro))
11.     print('macro: {}'.format(f1_macro))
12.     print('weighted: {}'.format(f1_weighted))
13.
14.     precision_micro = precision_score(y_true,y_pred,average='mi
    cro')
15.     precision_macro = precision_score(y_true,y_pred,average='ma
    cro')
```

```
16.     precision_weighted = precision_score(y_true,y_pred,average=
    'weighted')
17.
18.
19.     print('\nPrecision score')
20.     print('micro: {}'.format(precision_micro))
21.     print('macro: {}'.format(precision_macro))
22.     print('weighted: {}'.format(precision_weighted))
23.
24.     recall_micro = recall_score(y_true,y_pred,average='micro')
25.     recall_macro = recall_score(y_true,y_pred,average='macro')
26.     recall_weighted = recall_score(y_true,y_pred,average='weigh
    ted')
27.
28.     print('\nRecall score')
29.     print('micro: {}'.format(recall_micro))
30.     print('macro: {}'.format(recall_macro))
31.     print('weighted: {}'.format(recall_weighted))
32.
33.     acc = accuracy_score(y_true,y_pred)
34.
35.     print('\nAccuracy score: {}'.format(acc))
36.
37.     results = dict(
38.         f1_micro=f1_micro,
39.         f1_macro=f1_macro,
40.         f1_weighted=f1_weighted,
41.         precision_micro=precision_micro,
42.         precision_macro=precision_macro,
43.         precision_weighted=precision_weighted,
44.         recall_micro=recall_micro,
45.         recall_macro=recall_macro,
46.         recall_weighted=recall_weighted,
47.         accuracy=acc
48.     )
49.
50.     with open(f'tmp/{self.experiment_name}/result_{self.experim
    ent_name}_{class_scenario}.json', 'w') as fp:
51.         json.dump(results, fp)
52.
53.     return results
```

Kode 5.40 Fungsi Evaluation

Fungsi *evaluation* yang telah dihasilkan pada kode 5.40 digunakan untuk menghitung beberapa *metrics*, diantaranya adalah *f-score*, *precision*, *recall* dan *accuracy* yang diperoleh dari hasil klasifikasi oleh *learner*.

```
1.  def plot_confusion_matrix(self, cm, classes,
2.                            normalize=False,
3.                            title='Confusion matrix',
4.                            cmap=plt.cm.Blues):
5.
6.      plt.imshow(cm, interpolation='nearest', cmap=cmap)
7.      plt.title(title)
8.      plt.colorbar()
9.      tick_marks = np.arange(len(classes))
10.     plt.xticks(tick_marks, classes, rotation=45)
11.     plt.yticks(tick_marks, classes)
12.
13.     fmt = '.2f' if normalize else 'd'
14.     thresh = cm.max() / 2.
15.     for i, j in itertools.product(range(cm.shape[0]), range(cm.sha
    pe[1])):
16.         plt.text(j, i, format(cm[i, j], fmt),
17.                  horizontalalignment="center",
18.                  color="white" if cm[i, j] > thresh else "black")
19.
20.     plt.ylabel('True label')
21.     plt.xlabel('Predicted label')
22.     plt.tight_layout()
```

Kode 5.41 Fungsi Plot Confusion Matrix

Fungsi *plot_confusion_matrix* pada kode 5.41 digunakna untuk pembuatan *confusion matrix* yang menampilkan hasil prediksi dari *learner*.

```
1.  def save_confusion_matrix(self,y_true,y_pred,class_names,clas_scena
    rio):
2.      # Compute confusion matrix
3.      cnf_matrix = confusion_matrix(y_true, y_pred)
4.      # Plot non-normalized confusion matrix
5.      self.plot_confusion_matrix(cnf_matrix, classes=class_names, tit
    le='Confusion matrix')
6.      cm_name = self.fig_dir / f'{self.experiment_name}_{clas_scenari
    o}_confusionmatrix.png'
```

```
7.      plt.gcf().savefig(cm_name)
8.      #plt.show()
9.      plt.close()
```

Kode 5.42 Fungsi Save Confusion Matrix

Pada potongan kode 5.42, terdapat fungsi *save_confusion_matrix* yang digunakan untuk menyimpan hasil *confusion matrix* yang dihasilkan oleh fungsi *plot_confusion_matrix.*

```
1.   def classify(self):
2.       data_clas = TextClasDataBunch.load(path=Path(self.data_dir/sel
     f.dataset),cache_name=Path(f'tmp/clas/{self.experiment_name}'))
3.
4.       learn = text_classifier_learner(
5.           data_clas,
6.           bptt=self.bptt,
7.           pad_token=self.pad_token,
8.           path=self.model_dir.parent,
9.           model_dir=self.model_dir.name,
10.          emb_sz=self.emb_sz,
11.          nh=self.nh,
12.          nl=self.nl,
13.          drop_mult=self.clas_drop_mult,
14.          )
15.
16.      y_true, y_pred = self.classify_full(learn)
17.
18.      results = self.evaluation(y_true,y_pred, "full")
19.      class_names = data_clas.valid_ds.y.classes
20.      self.save_confusion_matrix(y_true,y_pred,class_names, "scratch"
         )
```

Kode 5.43 Fungsi Classify

Fungsi *classify* pada kode 5.43 merupakan fungsi yang pertama kali perlu dijalankan pada kelas *ClassifyingTwitter*. Fungsi ini bertugas untuk memuat *TextClasDataBunch* dari data *twitter* yang sesuai *subtask* yang diperlukan. Selanjutnya, dibuatlah objek *classifier* dengan menggunakan *learner text_classifier_learner*. Objek tersebut pun di *tuning*, untuk

kemduian diambil hasil prediksinya dan dievaluasi menggunakan fungsi-fungsi yang dicantumkan sebelumnya.

Halaman ini sengaja dikosongkan

BAB VI
HASIL DAN PEMBAHASAN

6.1 Hasil Perolehan Data Media Sosial

Proses pengambilan data dengan *streaming* pada media sosial *twitter* yang dilaksanakan beberapa kali dari tanggal 25 Maret 2019 hingga 4 Mei 2018 menghasilkan sebanyak 3.895.662 untuk *dataset language model*. Data yang telah diperoleh kemudian memasuki proses *filtering* bahasa dan *pre-processing*. Pada proses *filtering* bahasa, data yang berhasil diperoleh sebanyak 3.225.736 data. Data kemudian mengalami prapemrosesan untuk menghapus karakter yang tidak diperlukan. Untuk menyesuaikan dengan pengerjaan pada GermEval dan dengan keterbatasan kapasitas GPU, maka data *twitter* dengan domain *general* yang digunakan pada langkah pembuatan *language model* adalah sebanyak 306.911 *tweet*. Data ini untuk kemudian disimpan kedalam *TextLMDataBunch* untuk diproses lebih lanjut.

6.2 Hasil Perolehan Data Wikipedia

Pada perolehan data dari Wikipedia, jumlah artikel yang diperoleh adalah sebanyak 412.788 dengan total token sebanyak 65.733.214, dan token unik sebanyak 354.409. Untuk memaksimalkan kinerja dari *ULMFiT*, maka *vocabulary* dari model dibatasi hingga 60.001 token. Setelah dilakukan pembagian, maka *dataset* akan mengalami *postprocess* dengan membatasi *vocabulary* pada *data valid* dan *data test*, serta mengubah data yang berupa angka dengan karakter titik.

6.3 Hasil Pembuatan Pretrained Model

A. Pembuatan Pretrained Model Pada Data Wikipedia

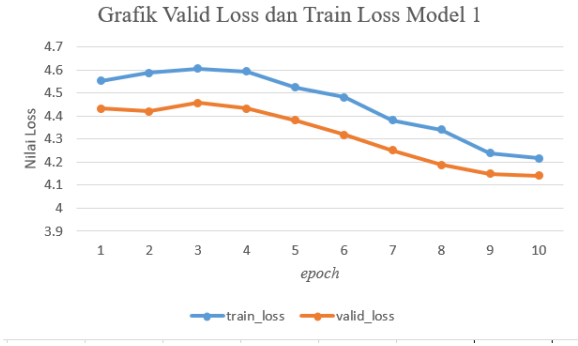
Pembuatan *pretrained model* pada data *Wikipedia* dimulai dengan penentuan parameter awal. Paremeter awal yang digunakan diambil dari skenario pada *github* FastAi. Parameter yang digunakan dalam pembuatan *pretrained model* ini dicantumkan dalam tabel 6.1.

Tabel 6.1 Tabel Parameter Awal *Pretrained Model* 1

No.	Parameter	Nilai
1	*Embedding size*	400
2	*Activation*	1150

3	*Layer*	3
4	*Batch size*	32
5	*BPTT*	70
6	*Epoch*	10
7	*Weight decay*	1e-7
8	*Learning rate*	5e-3
9	*Optimizer*	Adam, betas=(0.8, 0.99)
10	*Momentum*	(0.8,0.7)
11	*Clip*	0.12
12	*Dropout*	[0.25, 0.1, 0.2, 0.02, 0.15]
13	*Dps*	0.1

Proses pembuatan *pretrained model* ini menghabiskan waktu ± 100 menit untuk setiap *epoch*. Sehingga total dari waktu yang diperlukan untuk proses *training* model ini adalah 16.67 jam. Hasil dari proses *training model* berupa *train loss* dan *valid loss* dicantumkan pada gambar 6.1. Akurasi terbaik yang diperoleh dari *pretrained model* ini adalah sebesar 0.299747.



Gambar 6.1 Grafik *Train loss* dan *Valid loss* Model 1

Pada grafik 6.1, dapat dilihat bahwa nilai *train loss* dan *valid loss* memiliki *gap* yang kecil, namun *train loss* memiliki nilai lebih besar. Hal ini mengindikasikan adanya sedikit *underfitting* pada model dan diperlukan pelatihan yang lebih.

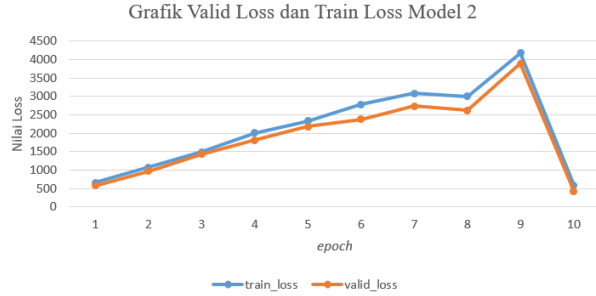
Selain menggunakan parameter tersebut, digunakan pula parameter yang diperoleh dari penggunaan ULMFiT pada

GermEval. Pada tabel 6.2, diperlihatkan parameter-parameter yang digunakan pada *pretrained model* yang kedua.

Tabel 6.2 Parameter *Pretrained Model* 2

No.	Parameter	Nilai
1	*Embedding size*	400
2	*Activation*	1150
3	*Layer*	3
4	*Batch size*	20
5	*BPTT*	70
6	*Epoch*	10
7	*Weight decay*	1e-7
8	*Learning rate*	3
9	*Optimizer*	SGD
10	*Momentum*	(0.95,0.85)
11	*Dropout*	[0.25, 0.1, 0.2, 0.02, 0.15]
12	*Dps*	0.5
13	*Pct_start*	0.9
14	*Div_factor*	10

Proses pembuatan *pretrained model* yang kedua ini menghabiskan waktu lebih panjang daripada model yang pertama, dengan waktu tiap *epoch* ± 120 menit dan total waktu yang dibutuhkan untuk menjalankan proses sebanyak 10 *epoch* adalah 20 jam. Hal ini dikarenakan nilai *batch size* yang lebih kecil daripada model sebelumnya. Hasil dari proses *training model* dicantumkan pada tabel 6.2.



Gambar 6.2 Grafik *Valid loss* dan *Train loss* Model 2

Pada gambar tersebut, terdapat nilai *loss* yang terlalu besar. Waktu yang dibutuhkan model pertama lebih cepat daripada model kedua, dengan perbedaan waktu ± 3.5 jam. Perbedaan tersebut dimungkinkan karena ketidaksesuaian parameter-parameter yang digunakan pada data. Sedangkan untuk performa, model pertama juga memiliki keunggulan yang lebih baik. Tabel 6.3 merupakan gambaran mengenai perbandingan akurasi dari model pertama dan kedua.

Tabel 6.3 Perbandingan Akurasi Model 1 & 2

Epoch	Model 1	Model 2
1	0.274	0.010
2	0.274	0.042
3	0.271	0.033
4	0.273	0.008
5	0.277	0.000
6	0.283	0.011
7	0.288	0.011
8	0.295	0.018
9	0.299	0.014
10	0.300	0.006

Dengan ini, dapat disimpulkan bahwa pada proses selanjutnya, akan digunakan model pertama dengan parameter mengikuti penelitian dari FastAi sebagai *pretrained model* pada tahapan berikutnya, yang selanjutnya disebut dengan *id-100*.

B. Pembuatan Pretrained Model Pada Data Twitter

Selain menggunakan *pretrained model* dengan memanfaatkan data *Wikipedia,* juga dilakukan eksperimen dengan menggunakan hasil *streaming* dan *pre-processing* data dari *twitter*. Parameter yang digunakan dalam pembuatan *pretrained model* dari data *twitter* ini sama dengan parameter untuk pembuatan model pertama dari data *Wikipedia*.

Waktu yang digunakan dalam proses *pretrain model* pada data *twitter* berlangsung ± 16 jam. Jika dibandingkan dengan model pertama pada data *Wikipedia, pretrained*

model pada data *twitter* memiliki akurasi dan *loss* yang lebih baik, seperti yang dicantumkan pada tabel 6.4.

Tabel 6.4 Perbandingan Akurasi dan Loss Model Wiki-100 dan Twitter

Model	Valid loss	Train loss	Akurasi (%)
id-100	4.216917	4.140793	29.97%
Twitter	2.727288	2.679495	59.3%

Meskipun memiliki performa yang lebih baik, pada langkah-langkah berikutnya, agar dapat menyesuaikan dengan penelitian sebelumnya, maka *pretrained model* yang digunakan tetap menggunakan model pertama.

6.4 Hasil Pelaksanaan *Fine-tuning Language Model*

6.4.1 Konfigurasi Parameter Awal

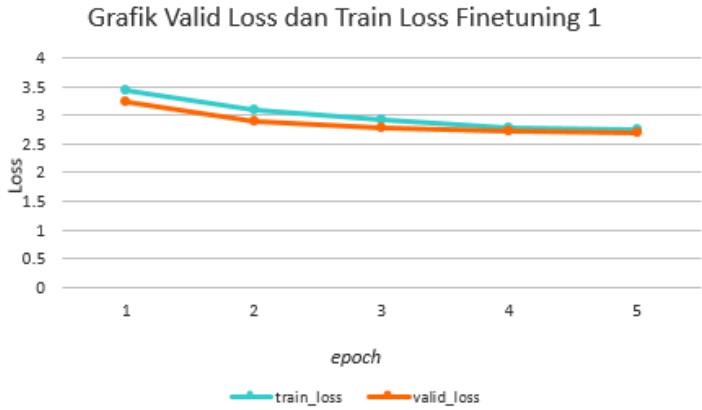
Pada pelaksanaan proses *fine-tuning* menggunakan *pretrained model* dan *language model Wikipedia*, parameter yang digunakan dicantumkan pada tabel 6.5. Parameter awal ini diperoleh dari penelitian sebelumnya.

Tabel 6.5 Parameter Awal Proses *Fine-tuning*

No.	Parameter	Nilai
1	Embedding size	400
2	Activation	1150
3	Layer	3
4	Batch size	64
5	BPTT	70
6	Epoch	5
7	Learning rate	0.01

6.4.2 Skenario *Full*

Pada skenario ini dilakukan proses *training* pada seluruh *layer* sekaligus. Proses dari *finetuning* ini dilakukan dengan melakkan proses *unfreezing* pada seluruh layer agar dapat dilakukan *training* secara bersamaan. Pada diagram 6.3, ditampilkan hasil berupa *train loss* dan *valid loss* dari proses *finetuning.*



Gambar 6.3 Grafik *Valid loss* dan *Train loss Finetuning 1*

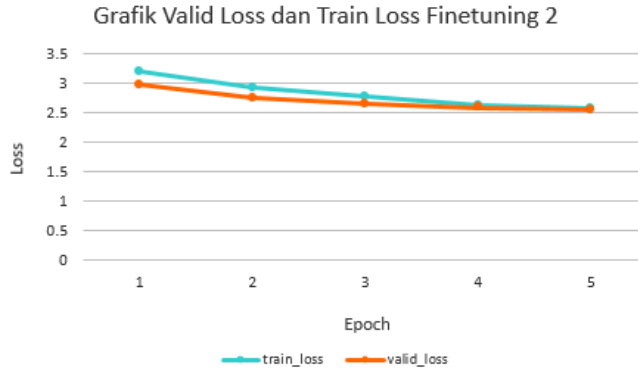
Pada grafik diatas, ditunjukkan bahwa nilai *valid loss* dan *train loss* memiliki jarak yang kecil, dengan nilai *train loss* lebih besar. Hal ini menunjukkan *underfitting* yang kecil, dimana menurut J. Howard masih diperlukan proses *training* yang lebih lanjut hingga *train loss* mampu turun dan *valid loss* dapat naik. Selain itu, dari data yang diambil dari proses *training*, dapat diperoleh kesimpulan bahwa hasil terbaik pada proses *fit* terdapat pada epoch kelima dengan akurasi 0.604575 atau 60.45%.

6.4.3 Skenario *Full + Discr*

Pada skenario ini, dilakukan proses *training* pada seluruh *layer* serta digunakan pula metode *discriminative learner rates*, dimana *learning rates* dari tiap *layer* memiliki ukuran yang berbeda. Untuk penelitian ini, ukuran *learning rates* yang digunakan tiap layernya adalah:

[lr/6, lr/3, lr, lr/2]

Komposisi perbandingan *learning rates* tersebut diperoleh dari penelitian J. Howard. Pada diagram 6.4, ditampilkan hasil dari *test loss* dan *valid loss* dari tiap *epoch*.

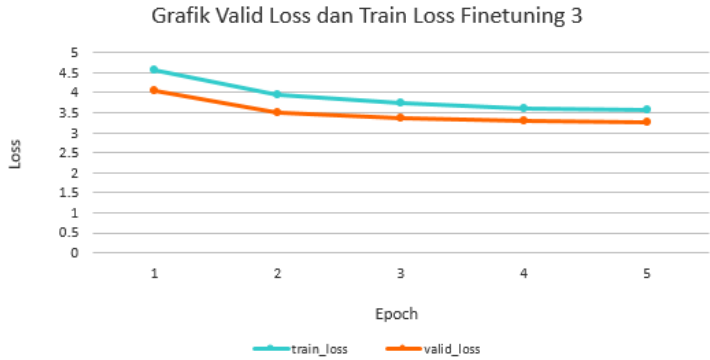


Gambar 6.4 Grafik *Valid loss* dan *Train loss Finetuning 2*

Dari gambar tersebut, dapat disimpulkan bahwa *train loss* dan *valid loss* yang dimiliki proses ini tidak terlampau jauh, namun masih mengindikasikan adanya *underfitting*. Melihat *train loss* yang semakin kecil bersamaan dengan *valid loss* yang juga semakin kecil, hal ini mengindikasikan model memiliki performa yang baik, hanya saja masih membutuhkan proses *training* lebih jauh. Dari data yang diambil dari proses *fine-tuning*, dapat diperoleh kesimpulan bahwa hasil terbaik pada proses *training* dari kelima *epoch* adalah pada *epoch* 5, dengan akurasi 0.62023 atau 62%. Angka ini lebih tinggi dari hasil terbaik pada *skenario* sebelumnya.

6.4.4 Skenario *Full + STLR*

Selain melakukan *tuning* pada seluruh *layer* sekaligus, pada skenario ini juga digunakan aturan *slanted triangular learning rates* menurut panduan dari *paper* ULMFiT, dimana *learning rates* meningkat secara cepat, yang kemudian turun secara perlahan. Pada penerapan ini, proses *training* akan dilakukan pada seluruh layer dengan dua fase perubahan.

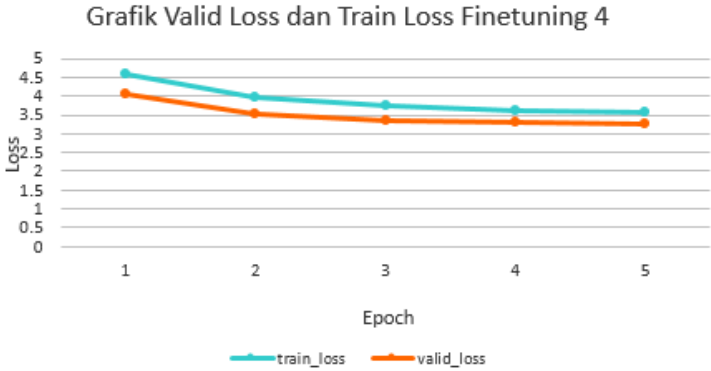


Gambar 6.5 Grafik *Valid loss* dan *Train loss Finetuning 3*

Grafik 6.5 menunjukkan dari nilai *valid loss* dan *train loss* dari tiap *epoch*. Dari grafik tersebut, dapat disimpulkan skenario ini membuat *language model* mengalami sedikit *underfitting*, dan untuk saat ini memiliki *loss* terbesar. Akurasi terbaik yang diperoleh dari proses ini adalah 0.521789 atau 52.17%.

6.4.5 Skenario *Full + Discr + STLR*

Pada skenario ini, dilakukan penggabungan terhadap tiga skenario sebelumnya, yaitu *full*, *discr*, dan *STLR*.



Gambar 6.6 Grafik *Valid loss* dan *Train loss Finetuning 4*

Pada grafik 6.6, dapat dilihat bahwa nilai *valid loss* memiliki kecenderungan untuk turun dan stagnan di angka 3 disaat *train loss* cenderung turun, hal ini mengindikasikan *underfitting* dan dibutuhkannya proses training lebih dari lima *epoch*. Untuk

akurasi terbaik yang dicapai oleh proses ini adalah 0.521789 atau 52%,

6.4.6 Perbandingan Seluruh Skenario

Pada tahapan *finetuning*, waktu yang dihabiskan untuk satu *epoch* rata-rata enam menit. Dengan itu, satu kali proses *fine-tuning* dengan satu skenario akan menghabiskan waktu sekitar 18 menit. Tabel 6.6 menunjukkan nilai *loss* dan akurasi terbaik dari masing-masing skenario *fine-tuning*.

Tabel 6.6 *Train loss, Valid loss,* dan Akurasi dari Language Model

Skenario	*Train loss*	*Valid loss*	*Accuracy (%)*
Full	2.74729	2.687508	60.4%
Full + Discr	2.565476	2.54444	62%
Full + STLR	3.564833	3.257825	52.17%
Full + Discr + STLR	3.564833	3.257825	51.17%

Berdasarkan tabel tersebut, dapat disimpulkan bahwa skenario ***finetuning full + discr*** memiliki akurasi terbaik dan akan digunakan untuk tahapan selanjutnya, yaitu pembuatan *classfier*.

6.5 Hasil Pembuatan *Classifier*

6.5.1 Konfigurasi Parameter Awal

Pada pelaksanaan proses pembuatan classifier menggunakan pretrained model *Wikipedia* yang telah megalami proses fine-tuning, parameter yang digunakan dicantumkan pada tabel 6.7. Parameter tersebut diperoleh dari penelitian sebelumnya.

Tabel 6.7 Parameter Awal Pembuatan Classifier

No.	Parameter	Nilai
1	*Embedding size*	400
2	*Activation*	1150
3	*Layer*	3
4	*Batch size*	64
5	*BPTT*	70

6	*Epoch*	15
7	*Learning rate*	0.001
8	*Learning rate (Discr)*	0.01
9	*Weight decay*	1e-7

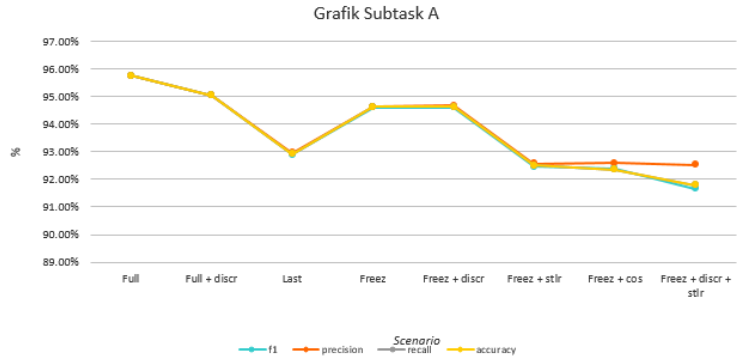
6.5.2 Subtask A

Pada *subtask A*, data yang digunakan pada pengklasifikasi ditunjukkan pada tabel 6.8.

Tabel 6.8 Jumlah Data *Subtask* A

Label	Jumlah Data
Positif	1,526
Negatif	2,009
Total	3,535

Setelah melaksanakan pembuatan *classifier* menggunakan 8 jenis skenario, pada tabel 6.7 dicantumkan hasil berupa *precision, recall, accuracy,* serta *f-measure* yang diperoleh dari delapan skenario tersebut.



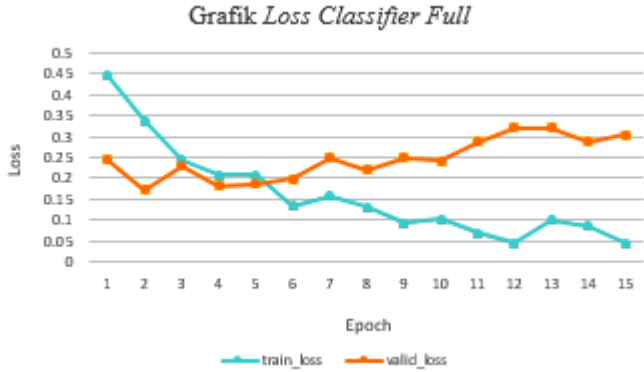
Gambar 6.7 Grafik *Precision, Recall, Accuracy* dan *F-Measure Subtask A*

Berdasarkan grafik tersebut, diperlihatkan bahwa performa terbaik dari proses diatas diperoleh pada skenario *full, full* +

discr, dan juga *freez discr*. Sedangkan performa model turun saat menggunakan metode *sltr* dan *cos*. Pada sub bab ini, akan dijelaskan mengenai performa dari tiga skenario terbaik tersebut.

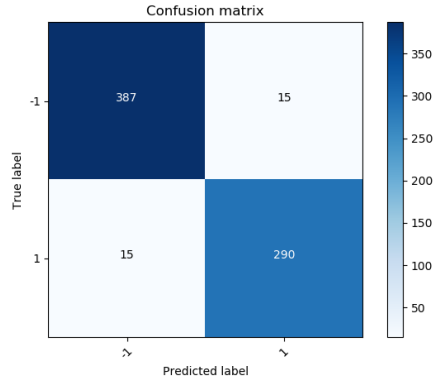
- **Skenario *Training Full***

Pada skenario ini, proses *training* dilakukan pada seluruh *layer* dari pengklasifikasi. Gambar 6.8 menunjukkan berapa *valid loss* dan *train loss* yang diperoleh pada tiap *epoch*.



Gambar 6.8 Grafik *Loss Classifier Full Subtask A*

Pada gambar 6.8, ditunjukkan bahwa *valid loss* memiliki kecenderungan untuk naik diatas *train loss*. Hal ini menunjukkan bahwa adanya *overfitting* dan sebaiknya proses *training* dilakukan pada *epoch* yang lebih kecil.



Gambar 6.9 *Confusion Matrix Full Subtask A*

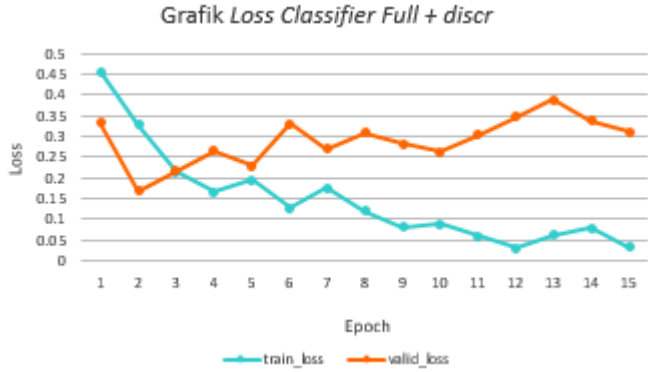
Gambar 6.9 merupakan hasil *confusion matrix* yang dihasilkan dari skenario *full* pada *subtask A.* Berdasarkan *confusion matrix* tersebut, dapat dilihat bahwa pengklasifikasi meiliki performa yang sangat baik.
Berikut merupakan

- **Skenario *Training Full + Discr***

Pada skenario ini, dilakukan *training* pada seluruh *layer* dengan menggunakan *discriminative learning rates*. Berikut merupakan komposisi *learning rates* yang digunakan pada skenario ini:

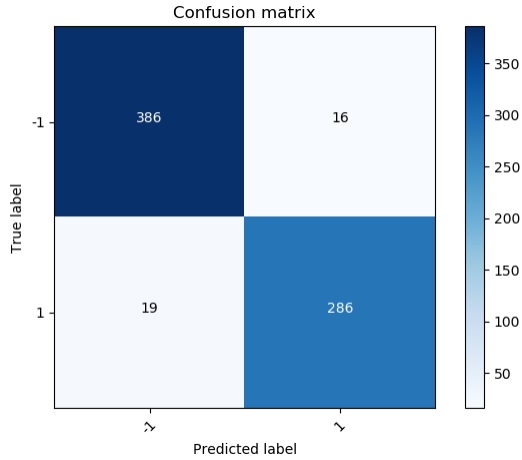
[lr/(lrm**4), lr/(lrm**3), lr/(lrm**2), lr/lrm, lr]

dimana lrm bernilai 2.6. Grafik *loss* skenario ini ditunjukkan pada gambar 6.10. Dapat dilihat dari gambar tersebut bahwa kondisi dari skenario ini tidak jauh berbeda, hanya saja memiliki nilai lebih kecil dari skenario *full.* Pada kasus ini, diperlukan ukuran *epoch* yang lebih kecil untuk mencapai hasil optimal.



Gambar 6.10 Grafik *Loss Classifier Full + discr Subtask A*

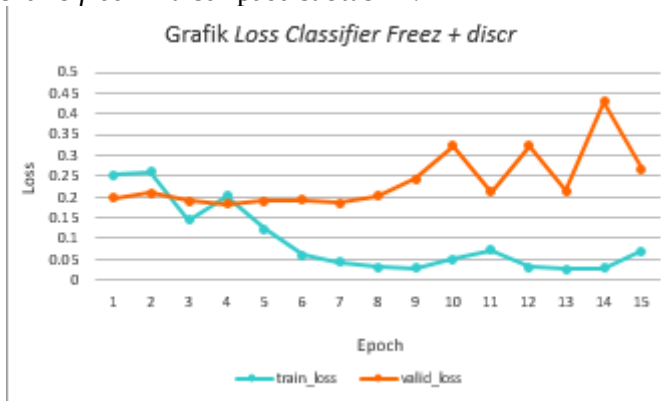
Sedangkan pada gambar 6.11, ditunjukkan *confusion matrix* yang dihasilkan dari skenario *full* dan *discr* pada *subtask A.* Dapat dilihat, skenario ini memiliki performa yang sangat baik dalam melakukan klasifikasi data berlabel.



Gambar 6.11 ***Confusion Matrix Full + discr Subtask A***

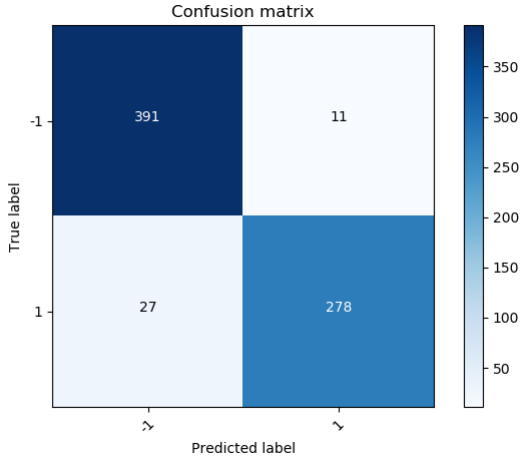
- **Skenario *Training Freez + Discr***

Pada skenario ini, dilakukan proses *training* secara bertahap pada setiap layernya dengan menggunakan fungsi *unfrozen*. Berikut merupakan hasil grafik *loss* yang dihasilkan dari skenario *freez + discr* pada *subtask A*.



Gambar 6.12 Grafik *Loss Classifier Freez + Discr Subtask A*

Pada grafik 6.12, dapat dilihat bahwa nilai *valid loss* akhir naik secara drastis disaat *train loss* terus turun bahkan mendekati nol, yang mengindikasikan adanya *overfitting*. Namun jika dilakukan *training* menggunakan *epoch* yang lebih kecil, misalnya dengan 5 *epoch*, hasil dari model akan lebih baik.



Gambar 6.13 *Confusion Matrix Freez Discr Subtask A*

Pada gambar 6.13, ditunjukkan *confusion matrix* dari skenario ini. Dari gambar tersebut, dapat disimpulkan bahwa skenario ini cukup memiliki performa yang baik.

- **Perbandingan Hasil Klasifikasi Tiga Skenario Terbaik Subtask A**

Untuk melakukan perbandingan atas hasil klasifikasi dari masing-masing skenario, pada tabel 6.9.

Tabel 6.9 Hasil Klasifikasi Subtask A

No.	Teks	*Full*	*Full Discr*	*Freez Discr*
1.	oke min ditunggu responnya	1	1	1
2.	akhirnya barang dari shopee nyampee, terimakasih atas pelayanannya yang baik shopee @shopeeid	1	1	1
3.	kenapa lama banget woy refund nya! mengecewakan	-1	-1	-1
4.	beli di tokped aja, mumpung ada diskon	-1	-1	-1

Pada tabel tersebut dapat dilihat bahwa hasil klasifikasi ketiga skenario menunjukkan hasil yang sama. Data nomor 1 dan 2 memiliki kelas positif (1) sedangkan data nomor 3 negatif (-1) dan nomor 4 netral. Karena pengklasifikasi hanya mampu melakukan klasifikasi ke dalam dua kelas, yaitu positif dan negatif, maka data nomor 4 dianggap sebagai data negatif oleh ketiga skenario tersebut.

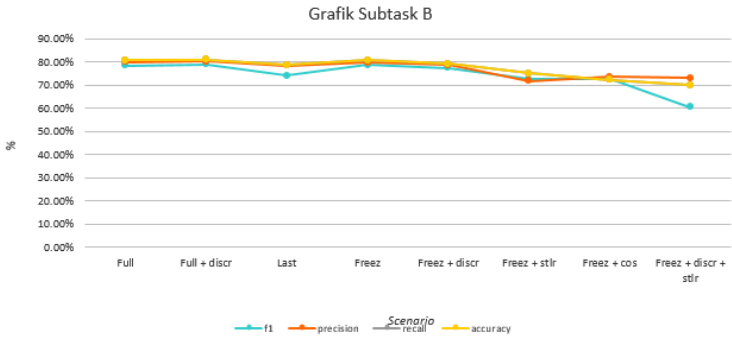
6.5.3 Subtask B

Pada *subtask B,* data yang digunakan sebagai dataset pada proses ini ditunjukkan pada tabel 6.10.

Tabel 6.10 Data dan Label pada Subtask B

Label	Jumlah Data
Positif	1,526
Netral	7,151
Negatif	2,009
Total	10,686

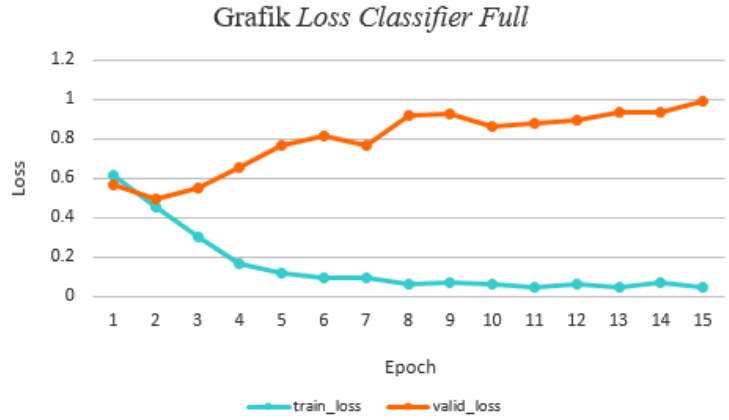
Setelah melaksanakan pembuatan *classifier* menggunakan 8 jenis skenario, pada grafik 6.14 dicantumkan hasil berupa *precision, recall, accuracy,* serta *f-measure* yang diperoleh dari delapan skenario tersebut. *Precision, recall,* dan *f-measure* tersebut diambil dari rata-rata masing-masing penghitungan tanpa pembobotan.



Gambar 6.14 Grafik *Precision, Recall, Accuracy,* dan *F-Measure Subtask B*

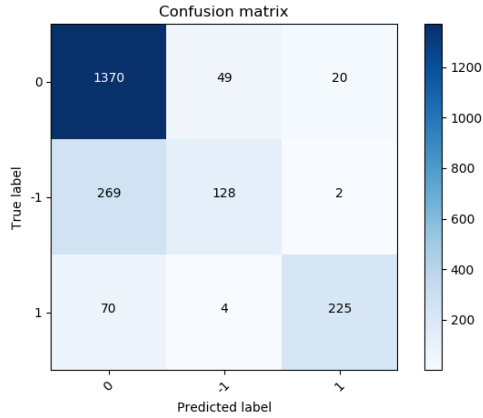
Pada grafik diatas, dapat dilihat bahwa performa terbaik didapatkan oleh skenario *full*, *full + discr*, dan *freez.* Sedangkan performa terburuk diperoleh oleh skenario dengan menggunakan *freez + discr + sltr*. Pada sub bab ini, akan dijelaskan mengenai ketiga skenario terbaik tersebut.

- **Skenario *Training Full***



Gambar 6.15 Grafik *Loss Classifier Full Subtask B*

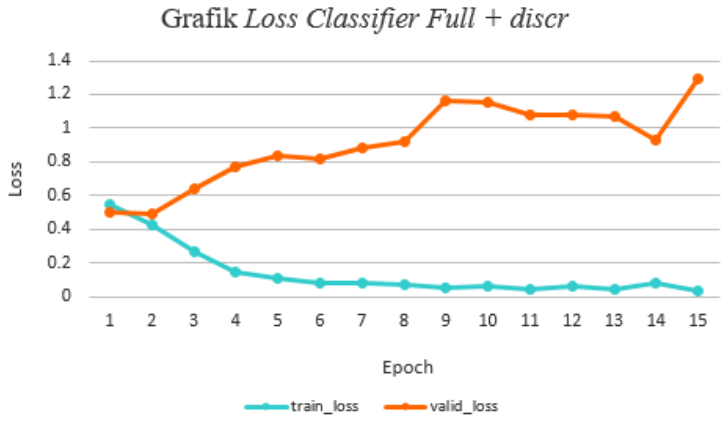
Pada gambar 6.15, diperlihatkan mengenai *loss* dari *valid loss* dan *train loss.* Dari grafik tersebut, dapat disimpulkan model sedikit mengalami *overfitting* karena semenjak *epoch* ke empat. Namun hal ini dapat ditanggulangi dengan mengurangi ukuran *epoch.*



Gambar 6.16 *Confusion Matrix Subtask B*

Pada gambar 6.16, diperlihatkan *confusion matrix* dari skenario *full* pada subtask B. Dari gambar tersebut, dapat disimpulkan bahwa model memiliki kemampuan yang baik dalam mengidentifikasi label netral. Namun, dapat dilihat bahwa cukup banyak data, yaitu 53 data, yang seharusnya berlabel negatif, namun masuk kedalam kelas netral. Sehingga, hal ini membuktikan presisi model yang belum cukup baik.

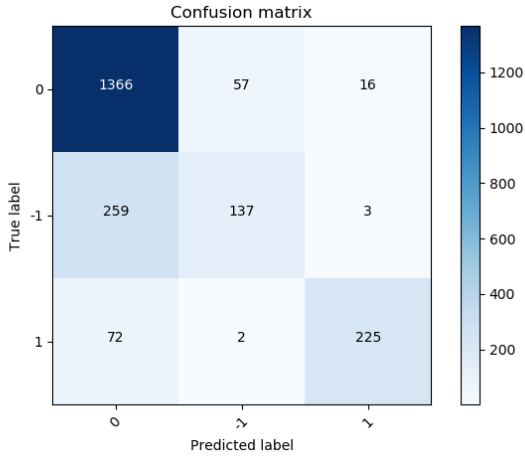
- **Skenario *Traininig Full + Discr***



Gambar 6.17 Grafik *Loss Classifier Full + discr*

Pada gambar 6.17, digambarkan mengenai hasil *loss* saat model mengalami *training*. Dari gambar tersebut, dapat disimpulkan

bahwa model tersebut telah memiliki *loss* yang cukup kecil, dengan ukuran *train loss* lebih kecil dari *valid loss* yang mengindikasikan adanya *overfitting*, terutama sejak *epoch* ke 4.

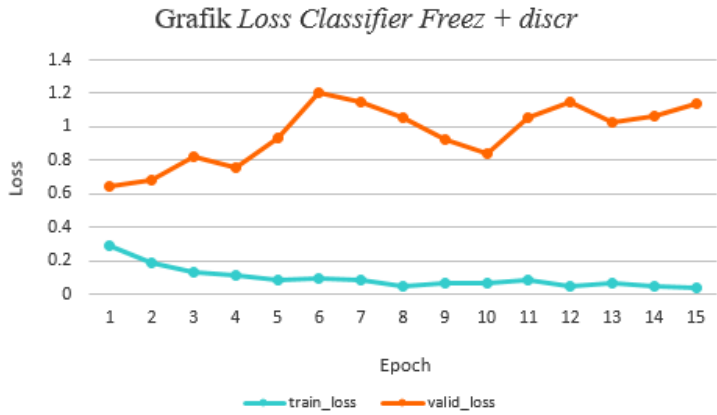


Gambar 6.18 *Confusion Matrix Full + discr Subtask B*

Gambar 6.18 menunjukkan *confusion matrix* dari penerapan skenario *full + discr* pada *subtask B.* Dari gambar itu dapat disimpulkan bahwa kemampuan model belum maksimal, terutama dalam menentukan data dengan label *negatif.*

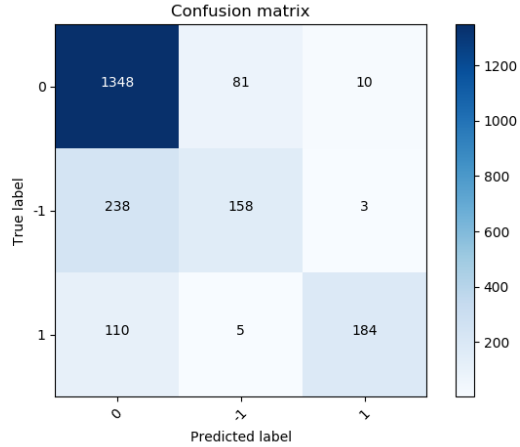
- **Skenario *Training Freez + Discr***

Skenario ini dilakukan dengan cara melakukan *training* pada masing-masing layer secara bergantian menggunakan fungsi *freeze*, seerta menggunakan *learning rates* yang berbeda tiap *layer*-nya apabila seluruh layer pada model telah sepenuhnya di-*unfreeze.*



Gambar 6.19 Grafik *Loss Classifier Freez + Discr Subtask B*

Pada grafik 6.19, ditunjukkan nilai *loss* dari skenario *freez* dan *discr*. Pada gambar tersebut dapat disimpulkan bahwa model memiliki *loss* yang cukup kecil, namun memiliki kecenderungan untuk *overfitting*. Hal tersebut dapat dihindari dengan pemilihan *epoch* yang lebih kecil,dan tidak jadi masalah apabila akurasi pada data validasi ikut membaik.



Gambar 6.20 *Confusion Matrix Freez + discr Subtask B*

Pada gambar 6.20, diperlihatkan bahwa model memiliki kemampuan yang baik dalam menilai data dengan label netral, namun kurang baik untuk model dengan label negatif maupun

positif. Hal ini pula yang menyebabkan nilai presisi dan *recall* yang lebih kecil.

- **Perbandingan Hasil Klasifikasi Tiga Skenario Terbaik Subtask B**

Untuk melakukan perbandingan atas hasil klasifikasi dari masing-masing skenario, pada tabel 6.11.

Tabel 6.11 Hasil Klasifikasi Subtask B

No.	Teks	*Full*	*Full Discr*	*Freez Discr*
1.	oke min ditunggu responnya	1	1	0
2.	akhirnya barang dari shopee nyampee, terimakasih atas pelayanannya yang baik shopee @shopeeid	0	1	1
3.	kenapa lama banget woy refund nya! mengecewakan	-1	-1	-1
4.	beli di tokped aja, mumpung ada diskon	0	0	0

Dari tabel diatas, dapat dilihat bahwa ketiga pengklasifikasi memberikan hasil yang beragam pada data nomor 1 dan 2. Data nomor 1 dan 2 memiliki kelas positif (1) sedangkan data nomor 3 negatif (-1) dan nomor 4 netral (0). Skenario yang berhasil memberikan prediksi paling tepat berdasarkan tabel diatas adalah *full discr*.

6.5.4 Subtask C

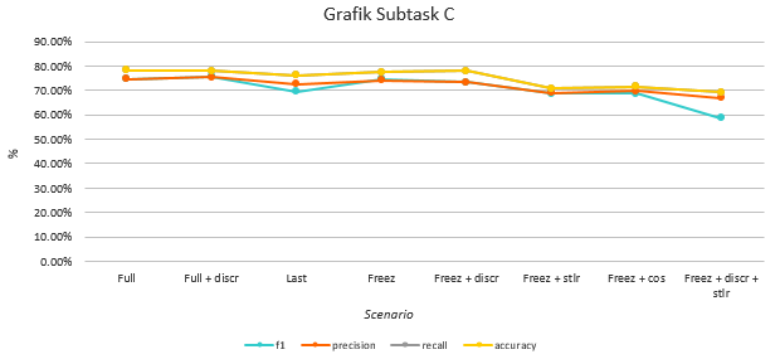
Pada *subtask C*, data yang digunakan pada pembuatan *classifier* ditunjukkan pada tabel 6.12.

Tabel 6.12 Data dan Label pada Subtask C

Label	Jumlah Data
Sangat Positif	33
Positif	1493
Netral	7151

Negatif	1553
Sangat Negatif	456
Total Data	10,686

Setelah melaksanakan pembuatan *classifier* menggunakan 8 jenis skenario yang telah disebutkan sebelumnya, pada tabel 6.21 dicantumkan hasil berupa *precision*, *recall*, *accuracy*, serta *f-measure* yang diperoleh dari delapan skenario tersebut.

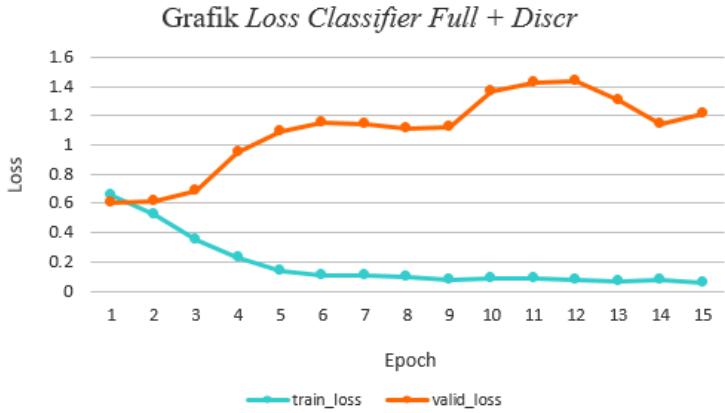


Gambar 6.21 Grafik *Precision, Recall, Accuracy,* dan *F-Measure* Subtask C

Pada *grafik* tersebut, dapat dilihat bahwa nilai *precision*, *recall*, *accuracy* dan *f-measure* pada subtask C lebih kecil daripada subtask yang lainnya. Nilai terbaik didapatkan oleh skenario *full discr*, *full discr*, serta *freez.* Sama dengan *subtask* lainnya, nilai performa paling kecil diperoleh skenario dengan menggunakan *stlr.*

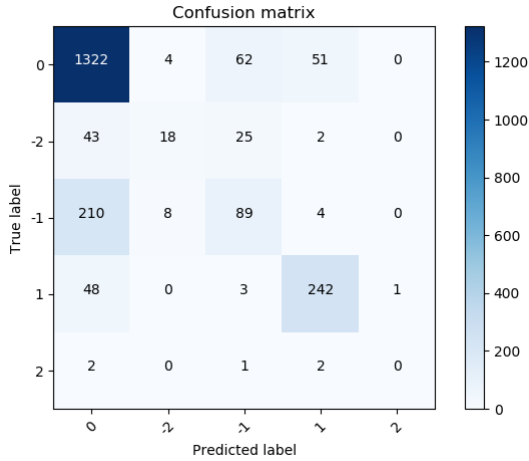
- **Skenario *Traininig Full + Discr***

Pada skenario ini, *training* dilaksanakan pada seluruh *layer* secara bersamaan dengan menggunakan *discriminative learning rates.*



Gambar 6.22 Grafik *Loss Classifier Full + discr Subtask C*

Pada gambar 6.22, dapat dilihat bahwa nilai *loss* dari *valid loass* memiliki kecenderungan naik, sedangkan *train loss* semakin turun hingga mendekati nol. Sama seperti grafik sebelumya, meskipun model memiliki kecenderungan untuk *overfitting*, hal ini dapat dicegah dengan menggunakan *epoch* yang lebih kecil.



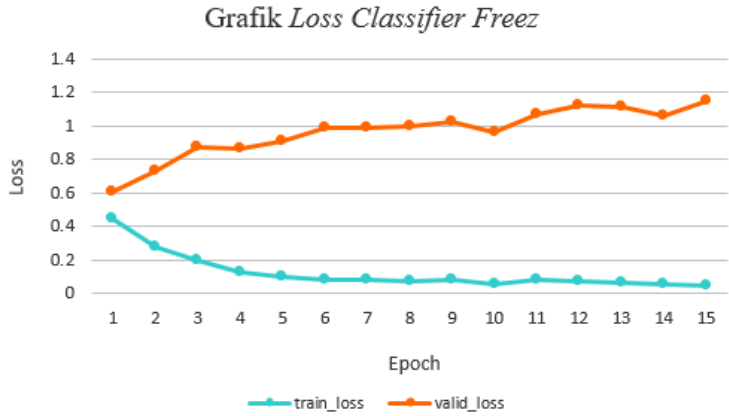
Gambar 6.23 *Confusion Matrix Full + discr Subtask C*

Confusion matrix pada gambar 6.23 menuntujukkan bahwa data meskipun telah cukup baik dalam mengenali data dengan label netral dan positif, namun model belum mampu mengeali data negatif, sangat negatif serta sangat positif dengan baik. Bahkan

model telah gagal dala mengidentifikasi data dengan label sangat positif hingga data yang diprediksi pada label tersebut hanya bernilai nol.

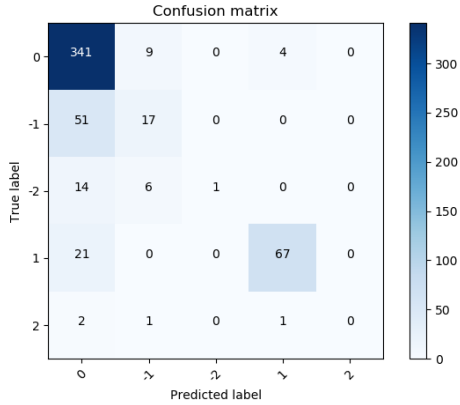
- **Skenario *Training Freez***

Skenario ini dijalankan dengan melakukan *training* secara bertahap pada tiap *layer* dengan mengunakan fungsi *freez*, hingga seluruh *layer* berhasil di *training.*



Gambar 6.24 Grafik *Loss Classifier Freez Subtask C*

Pada gambar 6.24, dapat dilihat bahwa *valid loss* memiliki kecenderungan untuk naik, sedangkan *train loss* turun. Masih sama pada grafik sebelumnya, meskipun ada kecenderungan untuk *overftitting*, namun hal ini dapat diatasi dengan *epoch* yang lebih kecil.

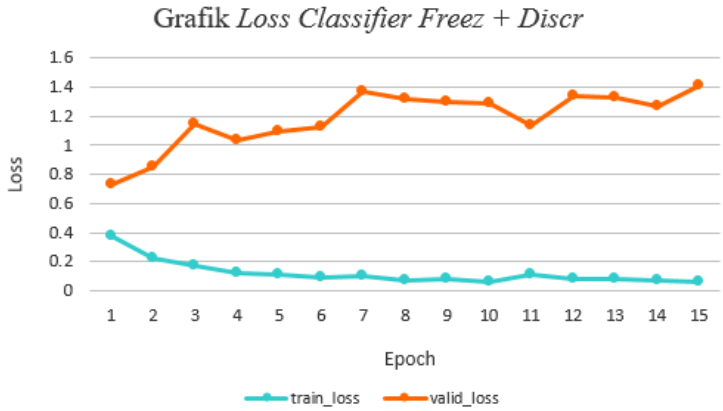


Gambar 6.25 *Confusion Matrix Freez Subtask C*

Pada *confusion matrix* yang ditunjukkan pada gambar 25, dapat dilihat bahwa model mampu mengenali data berlabel netral dan positif cukup baik, namun sebaliknya, data dengan label sangat positif dan sangat negatif tidak mampu dikenali dengan maksimal. Hal ini pula yang menyebakan ukuran dari *recall* dan *precision* yang relatif rendah dibanding *subtask* lain.

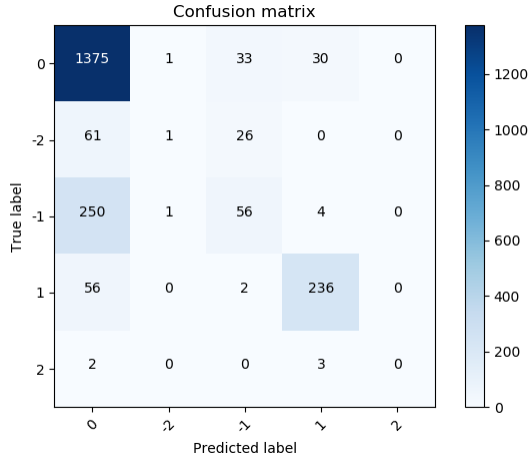
- **Skenario *Training Freez + Discr***

Pada skenario ini, dilakukan proses *training* dengan skenario *freez* dan *discriminative learning rates* pada pengaturan *learning rates*-nya.



Gambar 6.26 Grafik *Loss Freez + Discr Subtask C*

Pada grafik 6.26, dapat diperoleh hasil bahwa nilai *valid loss* memiliki kecenderungan naik daripada nilai *train loss*, yang lebih cenderung untuk turun. Meskipun menggambarkan adanya *overfitting*, namun hal ini dapat dicegah dengan penggunaan *epoch* yang lebih kecil.



Gambar 6.27 *Confusion Matrix Freez + Discr Subtask C*

Pada gambar 6.27 yang menunjukkan *confusion matrix* dari skenario ini, dapat dilihat kemampuan model yang belum cukup baik. Terutama dalam menilai data dengan label sangat positif dan sangat negatif.

- **Perbandingan Hasil Klasifikasi Tiga Skenario Terbaik Subtask C**

Untuk melakukan perbandingan atas hasil klasifikasi dari masing-masing skenario, pada tabel 6.13.

Tabel 6.13 Hasil Klasifikasi Subtask C

No.	Teks	*Full*	*Full Discr*	*Freez*
1.	oke min ditunggu responnya	1	1	1
2.	akhirnya barang dari shopee nyampee, terimakasih atas pelayanannya yang baik shopee @shopeeid	0	1	1

3.	kenapa lama banget woy refund nya! mengecewakan	-2	-1	-1
4.	beli di tokped aja, mumpung ada diskon	0	0	0
5.	pelayanan blibli emang luar biasa.. mana tampilan baruny jg keren bnget... semoga sukses selaluuu	1	0	-2
6.	kalo NGGAK NIAT BIKIN OLSHOP mending nggak usah bkin!!!! diskonny tipu tipu! nggak jelas!!! @bukalapak	-1	-1	0

Data nomor 1 dan 2 memiliki kelas positif (1) sedangkan data nomor 3 negatif (-1), nomor 4 netral, nomor 5 sangat positif (2), dan nomor 6 sangat negatif (-2). Dari tabel 6.13, dapat dilihat hasil yang diperoleh dari tiap skenario cukup beragam.

6.5.5 Pembahasan Hasil Percobaan Tiga Subtask

Pada tabel 6.14, digambarkan mengenai skenario terbaik beserta *subtask* dan hasil evaluasi yang diperoleh.

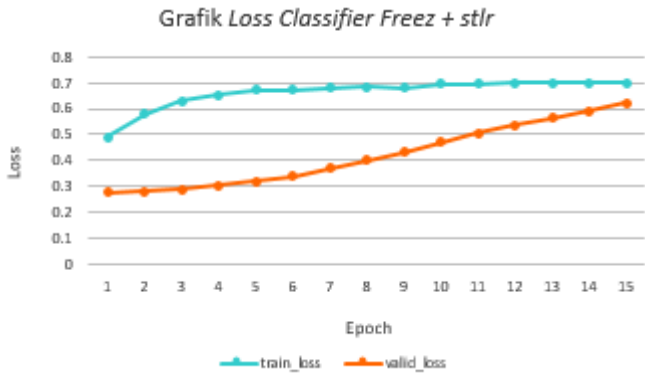
Tabel 6.14 Perbandingan Akurasi dan *F-measure* Seluruh Subtask

Subtask	Skenario	*F-measure (%)*	*Accuracy (%)*
Subtask A	*Full*	95.76%	95.76%
	Full Discr	95.05%	95.05%
	Freez Discr	94.61%	94.63%
Subtask B	*Full*	78.38%	80.63%
	Full Discr	78.86%	80.86%
	Freez Discr	78.55%	80.67%
Subtask C	*Full*	75.83%	78.19%
	Full Discr	74.73%	77.68%
	Freez	73.53%	78.05%

Dari tabel di atas, diperlihatkan bahwa skenario *training full layer* dengan tambahan pengaturan *discriminative fine-tuning* memiliki performa yang cukup baik pada seluruh *subtask.*

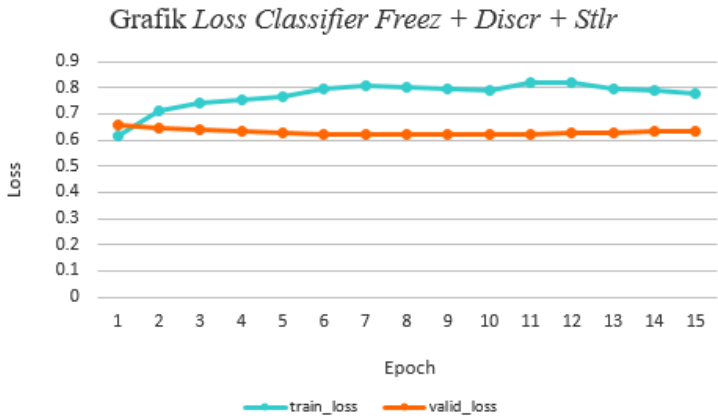
- **Pengaruh *Slanted Triangular Learning Rates***

Seperti yang dapat dilihat pada lampiran B hingga J, skenario yang menggunakan *slanted triangular learning rates* lebih cenderung untuk memiliki performa yang kecil.



Gambar 6.28 Grafik *Loss Classifier Freez + Stlr Subtask A*

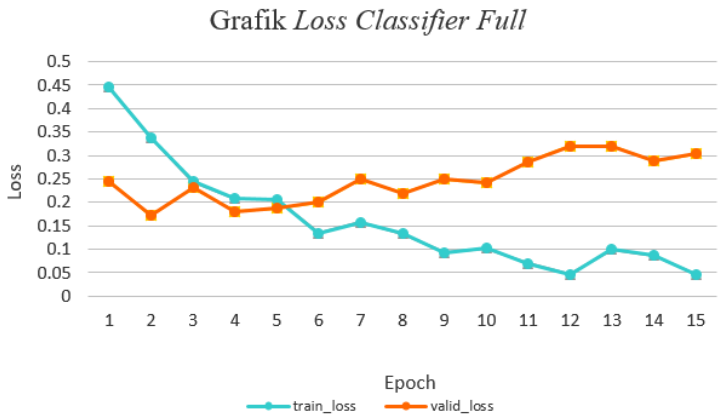
Bahkan pada skenario *freez* dan *stlr* di *subtask A*, seperti yang digambarkan pada grafik 6.28, dapat dilihat bahwa *training loss* memiliki ukuran yang lebih besar daripada *valid loss.* Hal ini menggambarkan adanya *underfitting* pada model. Selain ukuran *epoch, learning rates* juga dapat berpengaruh pada hasil dari skenario ini.



Gambar 6.29 Grafik *loss Freez + Discr + STLR* pada subtak B

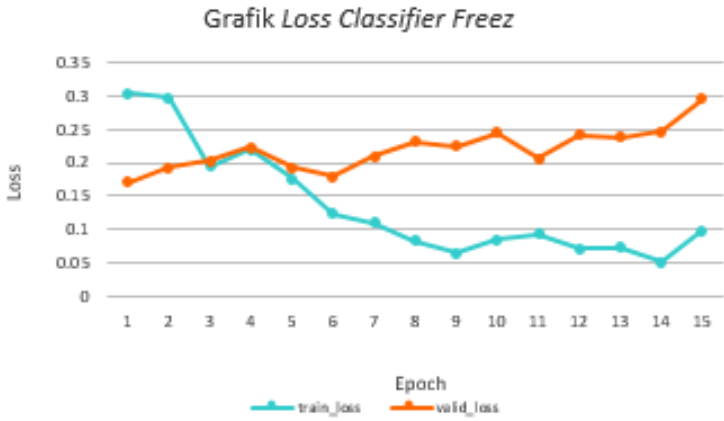
Keanehan juga muncul pada skenario *freez + discr + stlr* dimana justru terjadi adanya *underfitting* pada model. Normalnya, ukuran dari *valid loss* akan naik dan ukuran dari *train loss* akan turun seiring berjalannya proses *training* pada model. Pada gambar 6.29, dapat dilihat bahwa *train loss* justru naik disaat *valid loss* stagnan. Hal ini dapat terjadi akibat kurangnya data, ataupun model tidak mampu menyimpan *feature* dari data. Untuk lebih mengetahui pola dari *loss*, dibutuhkan *epoch* lebih banyak. Selain itu, dapat diterapkan solusi lain seperti menambah data berlabel untuk *classifier*, serta mengubah parameter-parameter yang ada.

- **Pengaruh *Gradual Unfreezing***



Gambar 6.30 Grafik *Loss Classifier Full*

Pada gambar 6.30, dapat dilihat grafik *loss* dari *classifier* dengan skenario *full* pada *subtask A.* Pada grafik tersebut diketahui bahwa *training loss* diperoleh sebesar 0.45, sedangkan *valid loss* 0.25 pada *epoch* pertama. Nilai *loss optimal* diperoleh pada *epoch* ke-6, dimana *train loss* mulai lebih kecil daripada *valid loss*. Hal ini dikarenakan pada saat *valid loss* lebih besar daripada *train loss*, hal ini menunjukkan model telah mempelajari *feature* dari data *training* dengan baik. Jika dilanjutkan pada *epoch* berikutnya, maka model akan mengalami *overfitting* dikarenakan *train losss* yang terlalu kecil dan *valid loss* yang semakin membesar. Dalam kata lain, model akan kehilangan kemampuan generalisasinya.

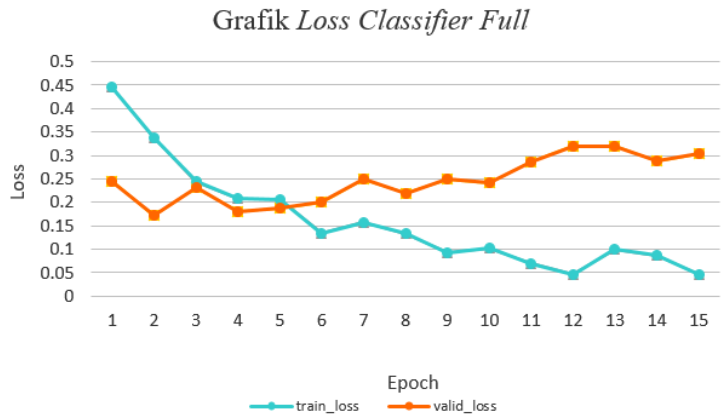


Gambar 6.31 Grafik *Loss Clasffier Freez*

Gambar 6.31 menunjukkan grafik *loss* dari *classifier* dengan skenario *freez* atau menggunakan *gradual unfreezing* pada *subtask A*, dimana *training* dilakukan tiap-tiap layer daripada dilakukan pada seluruh layer sekaligus. Pada gambar diatas diperoleh informasi bahwa *train loss* dan *valid loss* pada *epoch* pertama data mengalami penurunan dibanding dengan skenario *full*, yaitu dengan angka 0.3 pada *train loss* dan sekitar 0.16 pada *valid loss*. Selain itu, *loss* terbaik juga diperoleh lebih awal, yaitu pada *epoch 5*.

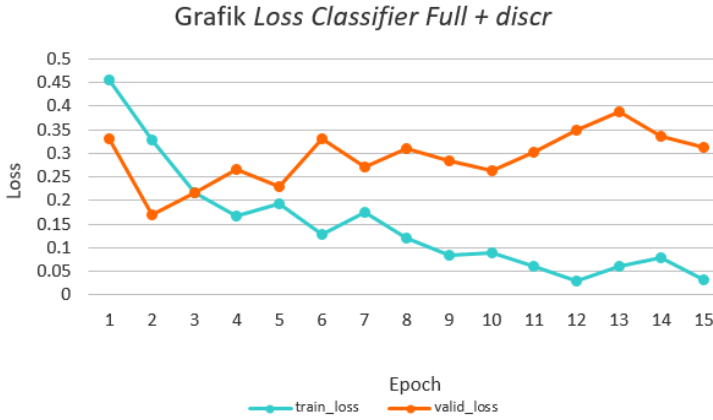
Dari uraian di atas dapat disimpulkan bahwa teknik *gradual unfreezing* mampu membantu model memperoleh hasil optimal lebih cepat dan dapat mengurangi ukuran dari *loss.*

- **Pengaruh *Discriminative Learning Rates***



Gambar 6.32 Grafik *Loss Classfier Full*

Pada gambar 6.32, dapat dilihat *loss* dari *classifier full* pada *subtask A.* Penjelasan mengenai grafik *loss* di atas dapat dilihat pada bagian sebelumnya.

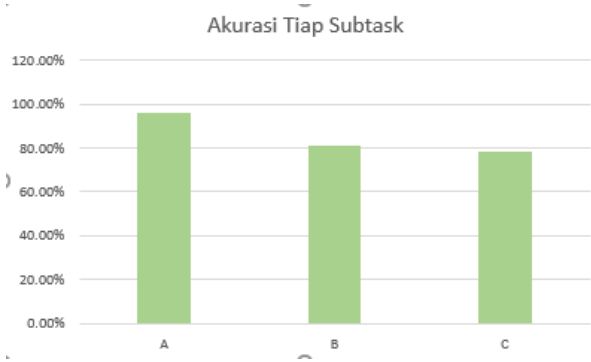


Gambar 6.33 Grafik *Loss Classifier Full + Discr*

Gambar 6.33 menunjukkan grafik *loss* pada *classifier* dengan skenario *full* pada *subtask A* ditambah dengan strategi *discriminative learning rates.* '*Discr*' merupakan suatu metode dimana setiap *layer* pada pengklasifikasi memiliki *learning rates* yang berbeda dikarenakan adanya kandungan informasi yang berbeda pula.

Pada grafik 6.33, dapat dilihat bahwa meskipun tidak merubah *loss* awal, namun *epoch* dengan *loss optimal* diperoleh lebih cepat, yaitu pada *epoch* ke-4. Dari penjelasan tersebut dapat disimpulkan bahwa pada penelitian ini, metode *discriminative learning rates* mampu memberikan hasil yang *optimal* dengan jumlah *epoch* yang lebih kecil.

- **Pengaruh Jumlah Data**



Gambar 6.34 Perbandingan Akurasi Terbaik Tiap Subtask

Selain itu pada tabel dapat dilihat bahwa, performa dari *Universal Language Model Fine-tuning* cenderung mengalami penurunan apabila kelas dari data bertambah, seperti yang digambarkan oleh gambar 6.34. Hal ini bisa disebabkan karena *data imbalance* ataupun hasil *tagging* yang kurang optimal.

BAB VII
KESIMPULAN DAN SARAN

Pada bab kesimpulan dan saran, akan dituliskan berbagai kesimpulan dari proses penelitian serta saran sehingga penelitian kedepannya dapat meningkatkan kualitasnya dibanding penelitian sebelumnya.

7.1 Kesimpulan

Kesimpulan dari tugas akhir analisis sentiment menggunakan *Universal Language Model Fine-tuning* ini adalah sebagai berikut:

1. Pada proses pembuatan *DataBunch* dilakukan pada tiga jenis data. Yang pertama untuk *pretrained model,* untuk *language model,* dan untuk *classifier.*
2. Pada proses pembuatan *pretrained model,* dibuat 3 jenis *pretrained model*. Model terbaik diperoleh pada *pretrained model* dari data *twitter.*
3. Pada proses *fine-tuning model,* terdapat empat jenis skenario, yaitu skenario *full, full + discr, full + stlr,* dan *full + discr + stlr*. Performa terbaik diperoleh pada skenario *full + discr.*
4. Proses pembuatan *classifier,* digunakan 8 jenis skenario. Yaitu skenario *full, full + discr, last, freez, freez + discr, freez + stlr, freez + cos,* dan *freez + discr + stlr.*
5. Pada klasifikasi pada dua jenis kelas, yaitu positif dan negatif, diperoleh akurasi terbaik 95.76% dengan skenario *full*, pada klasifikasi dengan tiga kelas, termasuk kelas netral, diperoleh akurasi 80.86% dengan skenario *full discr*, dan pada klasifikasi dengan lima kelas diperoleh akurasi 80.56% pada skenario *full.*
6. Skenario *full* ditambah dengan *discriminative learning rates* secara konsisten mampu masuk ke

dalam tiga besar skenario terbaik pada seluruh jenis klasifikasi.

7.2 Saran

Dalam pengerjaan tugas akhir, terdapat beberapa saran yang diharapkan dapat bermanfaat bagi pengembangan penelitian ke depan, yaitu:

1. Dibutuhkannya peningkatan kuantitas dan kualitas pelabelan dari data *classifier*, dikarenakan masih adanya kasus *underfitting* karena *data imbalance*, terutama pada data dengan lima kelas.
2. Perlu digunakannya hasil pretrained model menggunakan data twitter untuk pembuatan pengklasifikasi.
3. Pada skenario yang melibatkan *slanted triangular learning rates*, perlu adanya penyesuaian kembali dengan algoritma yang telah diperbaharui agar pengklasifikasi dapat memiliki performa yang lebih baik.
4. Dibutuhkan adanya perbandingan antara penggunaan GPU dengan CPU atau dengan spesifikasi GPU yang lebih tinggi.
5. Pada seluruh skenario, diperlukan adanya *tuning* parameter terbaik agar diberoleh hasil yang lebih memuaskan.

DAFTAR PUSTAKA

[1] A. Indonesia, "Infografis Penetrasi dan Perilaku Pengguna Internet Indonesia," *ID APJII*, 2017.

[2] B. Pang, *Opinion Mining and Sentiment Analysis*. .

[3] B. Pang and L. Lee, "Opinion Mining and Sentiment Analysis," *Found. Trends® InformatioPang, B., Lee, L. (2006). Opin. Min. Sentim. Anal. Found. Trends® Inf. Retrieval, 1(2), 91–231. doi10.1561/1500000001n Retr.*, vol. 1, no. 2, pp. 91–231, 2006.

[4] F. Neri, C. Aliprandi, and F. Capeci, "Sentiment Analysis on Social Media," *2012 IEEE/ACM Int. Conf. Adv. Soc. Networks Anal. Min.*, pp. 919–926, 2012.

[5] A. D. Putera, "Jumlah Pembeli 'Online' Indonesia Capai 11,9 Persen dari Populasi," *Kompas.com*, 2018. [Online]. Available: https://ekonomi.kompas.com/read/2018/09/07/164100326/jumlah-pembeli-online-indonesia-capai-119-pershttps://ekonomi.kompas.com/read/2018/09/07/164100326/jumlah-pembeli-online-indonesia-capai-119-persen-dari-populasien-dari-populasi. [Accessed: 04-Feb-2019].

[6] "Alexa Top 500 Global Sites." [Online]. Available: https://www.alexa.com/topsites.

[7] iPrice Group, "Peta E-Commerce Indonesia." [Online]. Available: https://iprice.co.id/insights/mapofecommerce/. [Accessed: 14-Mar-2019].

[8] J. Howard, "Universal Language Model Fine-tuning for Text Classification."

[9] M. Siegel, "Proceedings of the GermEval 2018 Workshop," 2018.

[10] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and Optimizing LSTM Language Models," 2015.

[11] S. Rosenthal, N. Farra, and P. Nakov, "SemEval-2017 Task 4 : Sentiment Analysis in Twitter," pp. 502–518, 2017.

[12] A. Pak and P. Paroubek, "Twitter as a corpus for

sentiment analysis and opinion mining.," in *LREc*, 2010, vol. 10, no. 2010.

[13] E. Alpaydin, *Introduction to machine learning*. MIT press, 2009.

[14] T. Signs, C. Bj, and R. N. Hansson, "Deep Learning Methods and Applications," 2017.

[15] A. M. Turing, "Computing machinery and intelligence," in *Parsing the Turing Test*, Springer, 2009, pp. 23–65.

[16] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[17] S. S. Haykin, *Neural networks and learning machines*, vol. 3. Pearson Upper Saddle River, 2009.

[18] E. D. Liddy, "Natural Language Processing," 2001.

[19] L. Y. Pratt, "Discriminability-based transfer between neural networks," in *Advances in neural information processing systems*, 1993, pp. 204–211.

[20] J. West, D. Ventura, and S. Warnick, "Spring research presentation: A theoretical foundation for inductive transfer," *Brigham Young Univ. Coll. Phys. Math. Sci.*, vol. 1, p. 32, 2007.

[21] S. Lai, K. Liu, S. He, and J. Zhao, "How to Generate a Good Word Embedding."

[22] A. Gladkova and A. Drozd, "Intrinsic Evaluations of Word Embeddings : What Can We Do Better ?," pp. 36–42, 2016.

[23] J. Guo, W. Che, H. Wang, and T. Liu, "Revisiting embedding features for simple semi-supervised learning," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 110–120.

[24] Y.-N. Chen, W. Y. Wang, and A. I. Rudnicky, "Leveraging frame semantics and distributional semantics for unsupervised semantic slot induction in spoken dialogue systems," in *Spoken Language Technology Workshop (SLT), 2014 IEEE*, 2014, pp. 584–589.

[25] D. Chen and C. Manning, "A fast and accurate dependency parser using neural networks," in

Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014, pp. 740–750.

[26] "About fast.ai," 2019. [Online]. Available: https://www.fast.ai/about/. [Accessed: 25-Feb-2019].

[27] S. Hochreiter and J. Schmidhuber, "LSTM can solve hard long time lag problems," in *Advances in neural information processing systems*, 1997, pp. 473–479.

[28] C. Olah, "Understanding lstm networks," 2015.

[29] A. M. Kaplan and M. Haenlein, "Users of the world, unite! The challenges and opportunities of Social Media," *Bus. Horiz.*, vol. 53, no. 1, pp. 59–68, 2010.

[30] W. K. Pertiwi, "Riset Ungkap Pola Pemakaian Medsos Orang Indonesia," 2018. [Online]. Available: https://tekno.kompas.com/read/2018/03/01/10340027/riset-ungkap-pola-pemakaian-medsos-orang-indonesia. [Accessed: 26-Feb-2018].

[31] "E Commerce," 2013. [Online]. Available: https://www.studymode.com/essays/e-Commerce-1554293.html. [Accessed: 26-Feb-2019].

[32] FastAi, "Log Loss."

[33] C. E. Metz, "Basic principles of ROC analysis," in *Seminars in nuclear medicine*, 1978, vol. 8, no. 4, pp. 283–298.

[34] Y. Sasaki, "The truth of the F-measure," *Teach Tutor mater*, vol. 1, no. 5, pp. 1–5, 2007.

[35] I. Loshchilov and F. Hutter, "Fixing weight decay regularization in adam," *arXiv Prepr. arXiv1711.05101*, 2017.

LAMPIRAN A: DATA KLASIFIKASI

- Contoh Data pada *Subtask A*

Label	***Tweet***
1	@LazadaIDCare @LazadaID Sip2 selalu puas belanja di @LazadaID
1	@TokopediaCare @gojekindonesia pesanan sudah diterima. terimkasih
1	@strwbrriys @womanfeeds Udah kok udah dikasih tau sama shopeecare ada alamatnya sendiri hahaha makasih banyak yaaaa
1	@TelkomCare melalui tokopedia, mandiri internet banking Terima Kasih
-1	@BukaBantuan Saya udah konfirmasi gak ada balasan sama sekali gimana ini dana saya
-1	@LazadaID saya butuh jawaban dan penanganan cepat, tolong jangan dijawab seperti ini lagi, bagaimana order an saya saat ini.
-1	@TokopediaCare iklan saya tidak tampil semua nya.. Mohon bantuan nya min
-1	@ShopeeCare koq ga di respon DM ny

- Contoh Data pada *Subtask B*

Label	*Tweet*
-1	@BukaBantuan barang pesanan saya belum sampai dari tgl 8 Mei 2018 sampai sekarang. Mohon bantuannya agar barang pesanan saya cepat sampai. Terimakasih kasih. Nb: No. Transaksi pesanan saya SBB : https://t.co/ooI5DsAkt9
-1	@TokopediaCare @ricormd Malam admin saya dah transfer ke BCA Tokopedia ko barang saya belum di kirim ya terimakasih
0	@geni_bos @LazadaIDCare Dugaan saya sih begitu. Kalau mau diteruskan, tolong saya di-update hasil akhirnya ya. Penasaran deh.
0	@irfantfk @LazadaID @LazadaIDCare @UnicomCareID Jaman now kaya gini ga perlu lagi bro.. Tinggal bawa ke service center aja. Nanti di cek via nomer IMEI.
1	Semua ada di @bukalapak termasuk paket internet murah, unlimited dan stabil
1	Semua keindahan membutuhkan proses #loveya @bliblidotcom

- Contoh Data pada *Subtask C*

Label	*Tweet*
-2	@BukaBantuan @diamondshop3 Eh min. Kalau ga ga berani kasih promo voucher mending ga usah. Dipikirin dulu konsekwensinya apa baru terapkan. SISTEM KALIAN ITU BODONG. DAN PEMAINNYA PIHAK BUKALAPAK SENDIRI
-2	@BlibliCare Gw minta CANCEL kenapa lo bilang pesanannya akan dikirim??!!! ngerti??!! @Kuzz00 @bliblidotcom @ferrysutanto
-1	@el_tandou @LazadaIDCare @LazadaID @YLKI_ID @TulusAbadi ini dananya lumayan besar... awalnya sy hanya mau ganti barang aj (tukar). ktanya barang kosong n ganti duit. udah kasih rekening ktanya 1-2 pekan. udah mau pekan ke 4 blm masuk2...
-1	@ShopeeCare Shopee kasih kabar dong soal proses pengembalian barang / dana saya, harus pembelanjaan di shopee mall kan dijamin ya, tp kok ini nga jelas status pengecekan brg nya sudah

	sampai dimana, jadi was was lo belanjanya
0	@BukaBantuan Ayo @Bukabantuan, mana penjelasannya? Saya harus tunggu berapa lama?
0	@ign_danang_ky @LazadaIDCare @xiaomiindonesia Beneran, bakal flash sale Redmi 5A tiap hari mulai 30 Januari 2018.
1	@BlibliCare @Dika40347703 ow begitu min, semoga dipermudah ya cara pengunaannya tks :)
1	@TokopediaCare Oke terima kasih min sudah diproses dengan segera permasalahan yang saya alami. Ditunggu progress selanjutnya min.
2	Big Thx @LazadaIDCare perubahan yg positif, tdk spt FS sblmnya, lama diterima. Kali ini cepat, 2 hr sdh ditangan.
2	Darii semua warna tampilan di @ShopeeID palinggg suka yg sekarang. :p

LAMPIRAN B: LOSS DAN AKURASI PETRAINED MODEL

- Model dengan parameter dari J. Howard

epoch	train_loss	valid_loss	accuracy
1	4.553018	4.431604	0.273705
2	4.587079	4.420712	0.274351
3	4.604301	4.456987	0.270731
4	4.593499	4.432752	0.273496
5	4.524287	4.380288	0.277323
6	4.479943	4.31714	0.282913
7	4.380641	4.251356	0.287832
8	4.340178	4.188244	0.294879
9	4.237709	4.14834	0.299186
10	4.216917	4.140793	0.299747

- Model dengan parameter dari GermEval

epoch	train_loss	valid_loss	accuracy
1	659.574585	573.113892	0.010267
2	1077.245972	966.112122	0.041919
3	1484.229736	1432.957397	0.032729
4	2011.552246	1811.932617	0.00824
5	2334.593018	2175.878174	0.000149
6	2783.844971	2378.986084	0.010752
7	3074.322266	2733.632812	0.011392
8	2995.286865	2616.998291	0.017679
9	4172.324219	3881.474365	0.013805
10	594.621033	431.701111	0.006416

- Model dengan *pretrained* data *Twitter*

epoch	train_loss	valid_loss	accuracy
1	3.258669	3.572228	0.51658
2	3.290778	3.097448	0.536058
3	3.341093	3.137618	0.529096
4	3.256202	3.029382	0.543938
5	3.153793	2.937925	0.556008
6	3.046445	2.855955	0.567321
7	2.877894	2.779376	0.577956
8	2.847057	2.714623	0.587179
9	2.767733	2.685712	0.591508
10	2.727288	2.679495	0.59309

LAMPIRAN C: LOSS DAN AKURASI FINETUNING LANGUAGE MODEL

- Skenario *Full*

epoch	train_loss	valid_loss	accuracy
1	3.444294	3.235111	0.538993
2	3.084488	2.906964	0.577073
3	2.927336	2.793858	0.590785
4	2.790057	2.728723	0.598926
5	2.74729	2.687508	0.604575

- Skenario *Full + Discr*

epoch	train_loss	valid_loss	accuracy
1	3.195221	2.978479	0.567323
2	2.909043	2.736868	0.596089
3	2.764027	2.642265	0.60806
4	2.626272	2.581449	0.615416
5	2.565476	2.54444	0.620232

- Skenario *Full + Discr + STLR*

epoch	train_loss	valid_loss	accuracy
1	4.572044	4.041984	0.426363
2	3.955159	3.517487	0.489138
3	3.750177	3.358684	0.508689
4	3.60286	3.285967	0.518427
5	3.564833	3.257825	0.521789

- Skenario *Full + STLR*

epoch	train_loss	valid_loss	accuracy
1	4.572044	4.041984	0.426363

2	3.955159	3.517487	0.489138
3	3.750177	3.358684	0.508689
4	3.60286	3.285967	0.518427
5	3.564833	3.257825	0.521789

LAMPIRAN D: LOSS DAN AKURASI CLASSIFIER SUBTASK A

- Skenario *Full*

epoch	train_loss	valid_loss	accuracy
1	0.444996	0.245282	0.934936
2	0.338511	0.171264	0.947666
3	0.244249	0.23066	0.930693
4	0.208112	0.180783	0.947666
5	0.20716	0.187122	0.950495
6	0.134547	0.199598	0.957567
7	0.15795	0.249756	0.947666
8	0.132736	0.219242	0.940594
9	0.093381	0.24864	0.950495
10	0.101731	0.241689	0.951909
11	0.069326	0.286544	0.950495
12	0.046395	0.320605	0.943423
13	0.099318	0.31982	0.93918
14	0.086167	0.288284	0.937765
15	0.046364	0.303105	0.943423

- Skenario *Full + Discr*

epoch	train_loss	valid_loss	accuracy
1	0.454922	0.331659	0.891089
2	0.32729	0.168622	0.944837
3	0.215599	0.216228	0.949081
4	0.166972	0.265601	0.943423
5	0.194235	0.228266	0.942008
6	0.127384	0.331309	0.940594
7	0.175788	0.270364	0.949081
8	0.119912	0.308698	0.944837

9	0.082623	0.282969	0.944837
10	0.089054	0.262277	0.950495
11	0.059624	0.302784	0.943423
12	0.030414	0.347913	0.950495
13	0.061455	0.387743	0.942008
14	0.079879	0.336568	0.944837
15	0.032717	0.311435	0.947666

- Skenario *Last*

epoch	train_loss	valid_loss	accuracy
1	0.492637	0.310162	0.922207
2	0.425438	0.258469	0.916549
3	0.392401	0.303009	0.87553
4	0.390873	0.220824	0.922207
5	0.404816	0.216281	0.923621
6	0.331338	0.206651	0.929279
7	0.362842	0.291548	0.908062
8	0.310684	0.246452	0.919378
9	0.272046	0.236726	0.923621
10	0.250159	0.276103	0.917963
11	0.223216	0.317369	0.915134
12	0.209573	0.317254	0.912306
13	0.223657	0.563502	0.872702
14	0.224234	0.406847	0.898161
15	0.200653	0.354095	0.916549

- Skenario *Freez*
  - Layer -1

epoch	train_loss	valid_loss	accuracy

1	0.492637	0.310162	0.922207

- Layer -2

epoch	train_loss	valid_loss	accuracy
1	0.409727	0.248247	0.923621

- Layer -3

epoch	train_loss	valid_loss	accuracy
1	0.353898	0.438349	0.796322

- Seluruh layer

epoch	train_loss	valid_loss	accuracy
1	0.304275	0.170858	0.946252
2	0.296784	0.191705	0.936351
3	0.193536	0.202446	0.947666
4	0.219659	0.222408	0.93918
5	0.175716	0.192121	0.946252
6	0.12376	0.179608	0.954738
7	0.110151	0.209099	0.954738
8	0.08297	0.231251	0.943423
9	0.064798	0.224239	0.949081
10	0.084683	0.245192	0.947666
11	0.09348	0.205484	0.950495
12	0.070566	0.242116	0.951909
13	0.072433	0.237365	0.937765
14	0.050136	0.246396	0.937765
15	0.097373	0.295915	0.933522

- Skenario *Freez + Discr*

epoch	train_loss	valid_loss	accuracy
1	0.502752	0.296597	0.923621

epoch	train_loss	valid_loss	accuracy
1	0.411401	0.223543	0.916549

epoch	train_loss	valid_loss	accuracy
1	0.315834	0.24385	0.905233

epoch	train_loss	valid_loss	accuracy
1	0.253459	0.196099	0.946252
2	0.260445	0.209088	0.937765
3	0.144814	0.190358	0.946252
4	0.20095	0.182486	0.947666
5	0.122207	0.19001	0.933522
6	0.059189	0.19345	0.944837
7	0.043501	0.185895	0.947666
8	0.029612	0.20293	0.949081
9	0.027203	0.242095	0.947666
10	0.050279	0.323329	0.920792
11	0.072199	0.212278	0.947666
12	0.031153	0.322978	0.937765
13	0.026138	0.214172	0.946252
14	0.028445	0.429871	0.930693
15	0.068611	0.267476	0.942008

- Skenario *Freez + STLR*

epoch	train_loss	valid_loss	accuracy
1	0.523284	0.292789	0.925035

epoch	train_loss	valid_loss	accuracy
1	0.657061	0.287224	0.91372

epoch	train_loss	valid_loss	accuracy
1	0.705001	0.272987	0.932108

epoch	train_loss	valid_loss	accuracy
1	0.488081	0.27397	0.925035
2	0.57889	0.279362	0.922207
3	0.627952	0.287925	0.91372
4	0.652202	0.302514	0.896747
5	0.67071	0.317388	0.882603
6	0.673116	0.339463	0.862801
7	0.67973	0.367471	0.833098
8	0.685711	0.401435	0.800566
9	0.681301	0.429961	0.793494
10	0.693889	0.468067	0.77652
11	0.693612	0.505732	0.745403
12	0.700599	0.537222	0.72843
13	0.699551	0.562013	0.70297
14	0.699453	0.589868	0.685997
15	0.697578	0.623431	0.680339

- Skenario *Freez + Cos*

epoch	train_loss	valid_loss	accuracy

1	0.502745	0.290882	0.920792
2	0.43864	0.250081	0.919378
3	0.398704	0.261909	0.905233
4	0.396679	0.217449	0.923621
5	0.399195	0.212508	0.922207
6	0.331061	0.20435	0.927864
7	0.357341	0.260205	0.916549
8	0.320631	0.291372	0.903819
9	0.287198	0.209164	0.925035
10	0.261991	0.251402	0.920792
11	0.238177	0.255997	0.922207
12	0.226449	0.253683	0.92645
13	0.233388	0.3849	0.899576
14	0.229904	0.491116	0.884017
15	0.206806	0.346734	0.91372

epoch	train_loss	valid_loss	accuracy
1	0.169605	0.239004	0.937765
2	0.193566	0.235249	0.92645
3	0.200405	0.237169	0.927864
4	0.191821	0.232557	0.929279
5	0.193305	0.229802	0.929279
6	0.189364	0.22651	0.932108
7	0.185414	0.224926	0.933522
8	0.184052	0.223171	0.934936
9	0.214893	0.226666	0.932108
10	0.207247	0.223177	0.936351
11	0.20645	0.221021	0.937765
12	0.199953	0.21854	0.937765
13	0.194389	0.217773	0.942008

14	0.19097	0.21342	0.942008
15	0.184559	0.212105	0.93918

epoch	train_loss	valid_loss	accuracy
1	0.344908	0.214339	0.937765
2	0.300319	0.210145	0.937765
3	0.2686	0.211498	0.936351
4	0.249009	0.213177	0.932108
5	0.235031	0.21401	0.933522
6	0.218172	0.219474	0.933522
7	0.204748	0.223433	0.932108
8	0.192332	0.224412	0.932108
9	0.18829	0.235032	0.929279
10	0.181778	0.239157	0.92645
11	0.2147	0.264056	0.923621
12	0.20806	0.285953	0.916549
13	0.205033	0.302652	0.909477
14	0.197345	0.301486	0.909477
15	0.19404	0.316343	0.903819

epoch	train_loss	valid_loss	accuracy
1	0.208535	0.258149	0.923621
2	0.203778	0.231149	0.932108
3	0.184414	0.216076	0.933522
4	0.167439	0.211798	0.937765
5	0.160236	0.215741	0.936351
6	0.153555	0.222063	0.933522
7	0.145227	0.222377	0.933522
8	0.140609	0.223441	0.933522
9	0.134149	0.221646	0.932108

10	0.136615	0.22298	0.929279
11	0.138672	0.226806	0.929279
12	0.13432	0.225494	0.927864
13	0.129302	0.22385	0.923621
14	0.146977	0.240986	0.92645
15	0.141631	0.2436	0.925035

- Skenario *Freez + Discr + STLR*

epoch	train_loss	valid_loss	accuracy
1	0.478869	0.303291	0.909477

epoch	train_loss	valid_loss	accuracy
1	0.715193	0.312217	0.906648

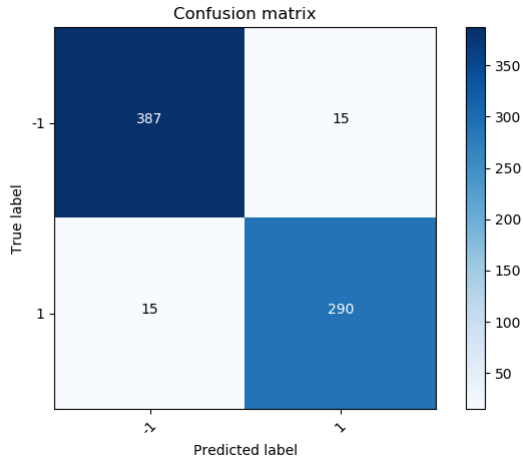
epoch	train_loss	valid_loss	accuracy
1	0.552008	0.288326	0.920792

epoch	train_loss	valid_loss	accuracy
1	0.420344	0.290038	0.917963
2	0.461325	0.271775	0.930693
3	0.48488	0.261988	0.937765
4	0.481304	0.257116	0.937765
5	0.480665	0.256065	0.934936
6	0.467312	0.257862	0.934936
7	0.462648	0.263558	0.925035
8	0.457642	0.269731	0.922207
9	0.446014	0.274513	0.915134
10	0.452802	0.283265	0.910891

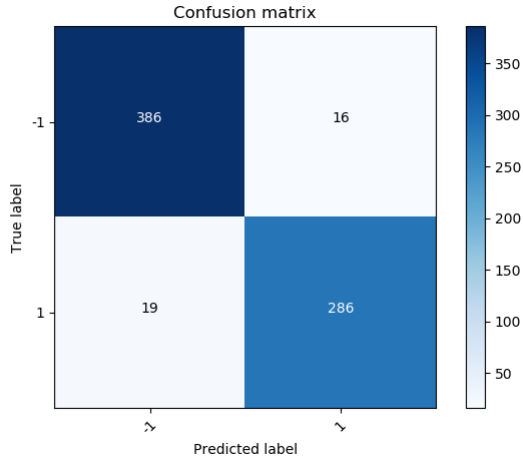
11	0.44579	0.290097	0.906648
12	0.44499	0.298535	0.896747
13	0.440539	0.305055	0.892504
14	0.435531	0.315021	0.869873
15	0.431112	0.320573	0.862801

LAMPIRAN E: CONFUSION MATRIX SUBTASK A

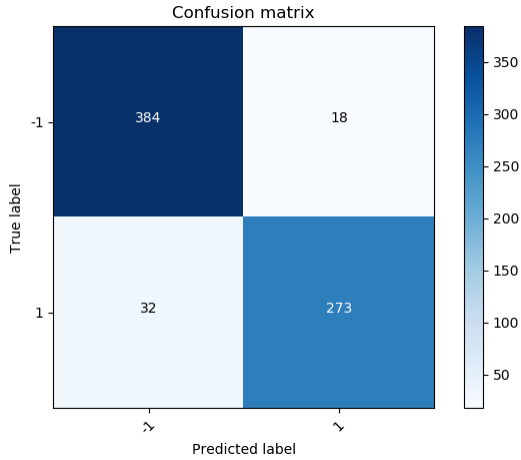
- Skenario *Full*



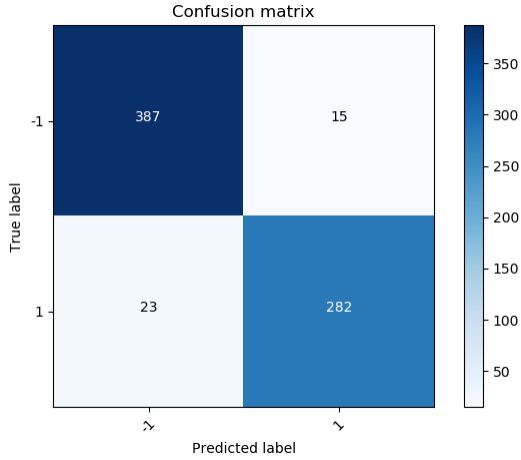
- Skenario *Full + Discr*



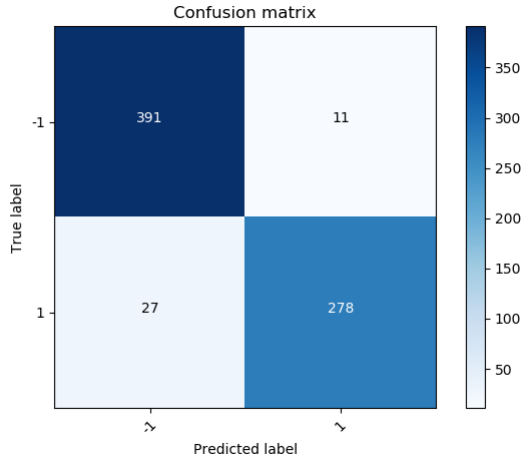
- Skenario *Last*



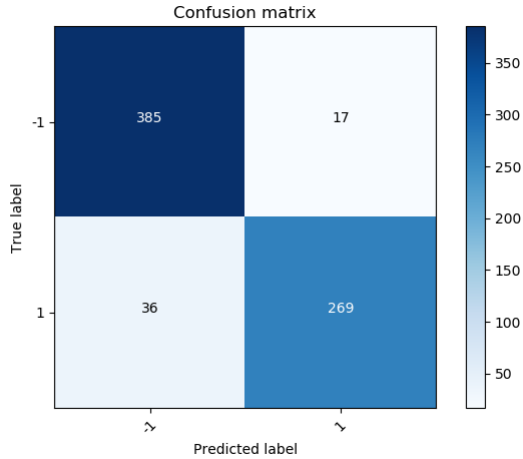
- Skenario *Freez*



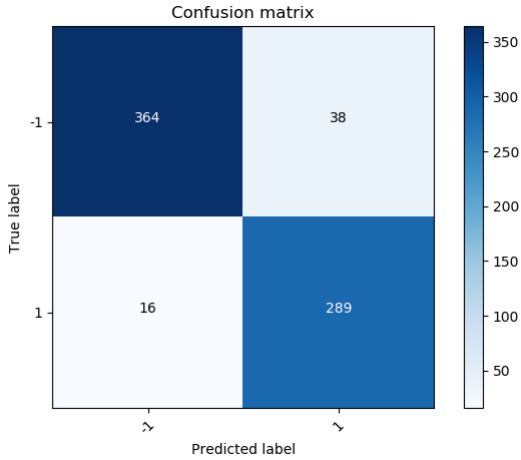
- Skenario *Freez + Discr*



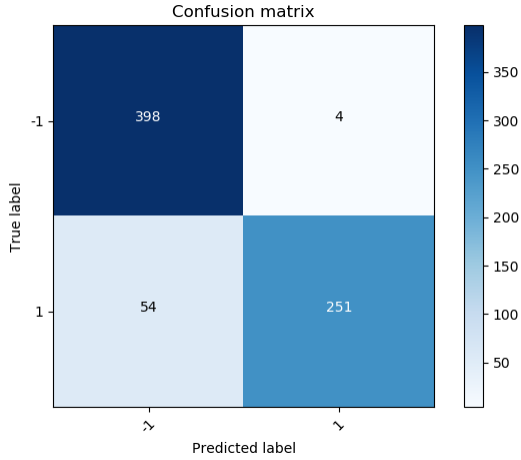
- Skenario *Freez + STLR*



- Skenario *Freez + Cos*



- Skenario *Freez + Discr + STLR*



LAMPIRAN F: LOSS & AKURASI CLASSIFIER SUBTASK B

- Skenario *Full*

epoch	train_loss	valid_loss	accuracy
1	0.614383	0.561751	0.791296
2	0.451146	0.492768	0.80627
3	0.29732	0.550871	0.799251
4	0.166753	0.655002	0.791296
5	0.116587	0.762112	0.787085
6	0.089254	0.808047	0.783341
7	0.090007	0.764274	0.803463
8	0.06178	0.913895	0.796444
9	0.065125	0.925012	0.799251
10	0.059166	0.857777	0.794572
11	0.044287	0.878044	0.791764
12	0.063471	0.893233	0.803931
13	0.044037	0.933817	0.788489
14	0.067417	0.931343	0.795508
15	0.046187	0.98514	0.800655

- Skenario *Full + Discr*

epoch	train_loss	valid_loss	accuracy
1	0.546515	0.501854	0.79504
2	0.422006	0.492465	0.80861
3	0.262927	0.636483	0.77679
4	0.14296	0.766719	0.789424
5	0.111945	0.832992	0.784745
6	0.080608	0.817518	0.788021
7	0.081457	0.883486	0.779598

8	0.073383	0.92	0.785681
9	0.055304	1.162827	0.77211
10	0.057897	1.151117	0.7927
11	0.040504	1.074469	0.79036
12	0.058253	1.074259	0.794572
13	0.043618	1.066765	0.783809
14	0.075908	0.924655	0.793168
15	0.033667	1.291543	0.796912

- Skenario *Last*

epoch	train_loss	valid_loss	accuracy
1	0.679831	0.636732	0.735143
2	0.568328	0.596412	0.756668
3	0.518766	0.610657	0.75854
4	0.504172	0.609926	0.766963
5	0.474456	0.592805	0.773982
6	0.488285	0.676686	0.752925
7	0.462151	0.639063	0.766963
8	0.440085	0.666144	0.765091
9	0.418798	0.688167	0.762284
10	0.411762	0.666309	0.770707
11	0.40168	0.692898	0.765559
12	0.398819	0.628295	0.781937
13	0.377152	0.612327	0.785681
14	0.38302	0.613859	0.768367
15	0.361904	0.644499	0.773982

- Skenario *Freez*

epoch	train_loss	valid_loss	accuracy

1	0.679831	0.636732	0.735143

epoch	train_loss	valid_loss	accuracy
1	0.559253	0.576215	0.771175

epoch	train_loss	valid_loss	accuracy
1	0.467006	0.509568	0.788489

epoch	train_loss	valid_loss	accuracy
1	0.361915	0.515835	0.806738
2	0.221539	0.623766	0.791296
3	0.142187	0.687727	0.776322
4	0.105704	0.716244	0.790828
5	0.090451	2.191536	0.700983
6	0.073075	0.958825	0.796912
7	0.06912	0.842201	0.785213
8	0.058757	0.851355	0.780066
9	0.064054	0.845981	0.760412
10	0.063729	0.767949	0.797847
11	0.063912	0.877555	0.792232
12	0.05228	0.88131	0.797379
13	0.05951	0.922858	0.781001
14	0.051725	0.898091	0.775386
15	0.045797	0.937464	0.794104

- Skenario *Freez + Discr*

epoch	train_loss	valid_loss	accuracy
1	0.602761	0.566567	0.770707

epoch	train_loss	valid_loss	accuracy
1	0.553966	0.531151	0.781469

epoch	train_loss	valid_loss	accuracy
1	0.452368	0.552305	0.787085

epoch	train_loss	valid_loss	accuracy
1	0.291998	0.641344	0.790828
2	0.18737	0.67776	0.790828
3	0.132982	0.824894	0.787553
4	0.110356	0.753744	0.793168
5	0.081318	0.934863	0.79504
6	0.089369	1.207766	0.782873
7	0.083402	1.149729	0.793636
8	0.047112	1.053683	0.79504
9	0.068252	0.92665	0.795508
10	0.067942	0.843701	0.803463
11	0.080756	1.050514	0.784745
12	0.047507	1.143677	0.783341
13	0.065499	1.025891	0.734207
14	0.044544	1.060873	0.780066
15	0.037134	1.136077	0.792232

- Skenario *Freez + STLR*

epoch	train_loss	valid_loss	accuracy
1	0.751184	0.683547	0.764623

epoch	train_loss	valid_loss	accuracy

1	0.762198	0.702662	0.751989

epoch	train_loss	valid_loss	accuracy
1	0.721383	0.697007	0.756668

epoch	train_loss	valid_loss	accuracy
1	0.663677	0.701814	0.750585
2	0.732773	0.7132	0.74497
3	0.76732	0.723173	0.729995
4	0.779787	0.74317	0.713149
5	0.807412	0.765542	0.694899
6	0.826973	0.780909	0.688816
7	0.839752	0.792881	0.679925
8	0.834858	0.806583	0.670566
9	0.826553	0.824189	0.655124
10	0.831832	0.834775	0.648105
11	0.85209	0.835759	0.645765
12	0.847793	0.849753	0.641554
13	0.837023	0.858871	0.633131
14	0.83957	0.857957	0.634534
15	0.835256	0.861107	0.631259

- Skenario *Freez + Cos*

epoch	train_loss	valid_loss	accuracy
1	0.663974	0.641254	0.72204
2	0.56939	0.594433	0.762284
3	0.516085	0.587809	0.759476
4	0.499107	0.593574	0.767899
5	0.472931	0.580851	0.773046

6	0.485925	0.667242	0.751989
7	0.453622	0.683066	0.741694
8	0.437787	0.5919	0.77211
9	0.419932	0.603616	0.770239
10	0.418266	0.652586	0.759008
11	0.396735	0.670283	0.757604
12	0.395065	0.612041	0.779598
13	0.383936	0.579059	0.787553
14	0.376467	0.608289	0.77445
15	0.377128	0.614385	0.777258

epoch	train_loss	valid_loss	accuracy
1	0.341028	0.615313	0.776322
2	0.382796	0.613942	0.777258
3	0.377341	0.610967	0.77679
4	0.472671	0.612051	0.775854
5	0.486752	0.609034	0.781469
6	0.489958	0.60523	0.778662
7	0.490847	0.600727	0.782873
8	0.472561	0.600515	0.779598
9	0.477695	0.604781	0.775854
10	0.46457	0.609638	0.773514
11	0.461029	0.612276	0.773982
12	0.449396	0.613686	0.77211
13	0.43787	0.617738	0.771175
14	0.424222	0.621935	0.770239
15	0.409444	0.615953	0.77211

epoch	train_loss	valid_loss	accuracy
1	0.373545	0.748299	0.664951

2	0.366942	0.741707	0.672438
3	0.375846	0.718433	0.686476
4	0.363823	0.688466	0.705194
5	0.3508	0.665864	0.719701
6	0.331341	0.655929	0.733271
7	0.312319	0.643244	0.736547
8	0.298951	0.636404	0.740758
9	0.29482	0.638426	0.73795
10	0.291532	0.63783	0.741694
11	0.283266	0.636871	0.743098
12	0.279345	0.639589	0.743098
13	0.274381	0.636713	0.748245
14	0.267004	0.634228	0.7562
15	0.260185	0.640695	0.756668

epoch	train_loss	valid_loss	accuracy
1	0.361174	0.668151	0.722508
2	0.321678	0.663412	0.736079
3	0.30191	0.625475	0.757604
4	0.276619	0.621745	0.766495
5	0.265676	0.628231	0.764623
6	0.254013	0.633242	0.76088
7	0.24302	0.640791	0.753861
8	0.231507	0.648823	0.748713
9	0.219567	0.657777	0.739822
10	0.209102	0.661808	0.743098
11	0.196848	0.668087	0.740758
12	0.193264	0.678644	0.736079
13	0.19744	0.717562	0.729995
14	0.19302	0.726291	0.733739

15	0.187683	0.738853	0.732803

- Skenario *Freez + Discr + STLR*

epoch	train_loss	valid_loss	accuracy
1	0.610557	0.64948	0.704258

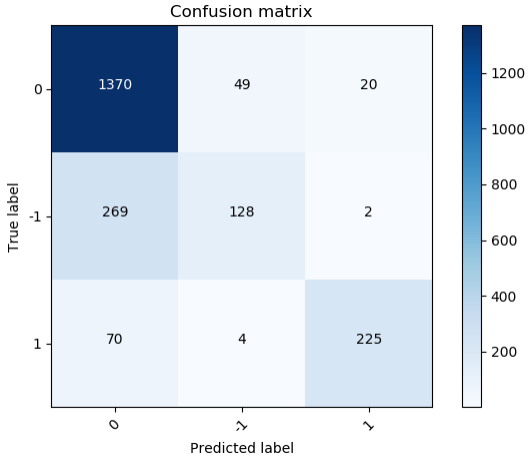
epoch	train_loss	valid_loss	accuracy
1	0.565538	0.648715	0.70613

epoch	train_loss	valid_loss	accuracy
1	0.597824	0.642728	0.708938

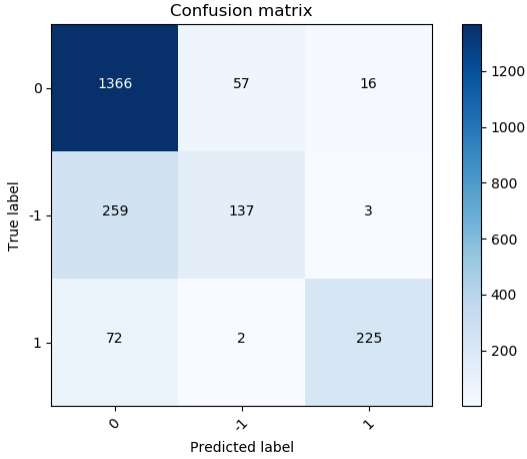
epoch	train_loss	valid_loss	accuracy
1	0.616902	0.654373	0.700515
2	0.713436	0.644849	0.709874
3	0.742297	0.637632	0.717829
4	0.750089	0.630405	0.727656
5	0.764964	0.626897	0.735143
6	0.797396	0.623513	0.743098
7	0.80557	0.621264	0.751521
8	0.802401	0.621084	0.754796
9	0.792917	0.62014	0.755732
10	0.789633	0.618567	0.754796
11	0.820439	0.620622	0.753393
12	0.816646	0.624487	0.751053
13	0.795275	0.627765	0.750117
14	0.787599	0.630592	0.746373
15	0.776033	0.634159	0.740758

LAMPIRAN G: CONFUSION MATRIX SUBTASK B

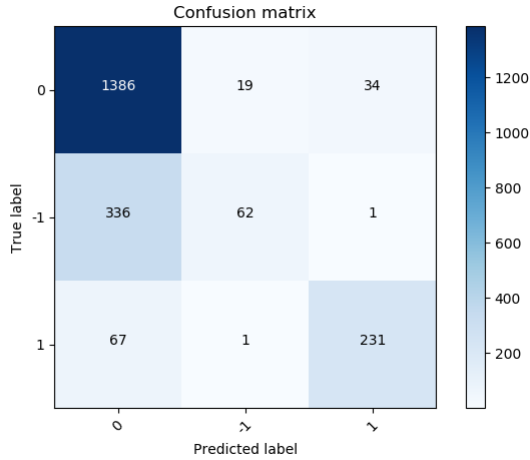
- Skenario *Full*



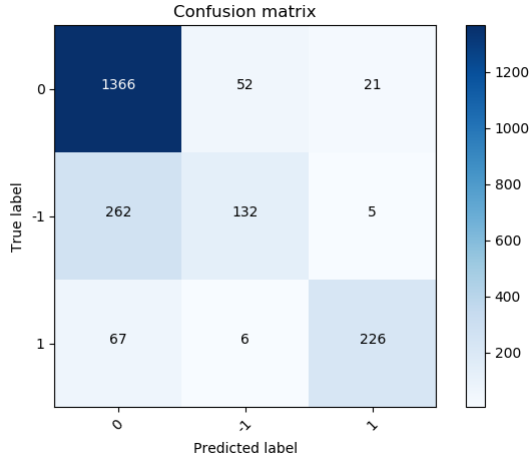
- Skenario *Full + Discr*



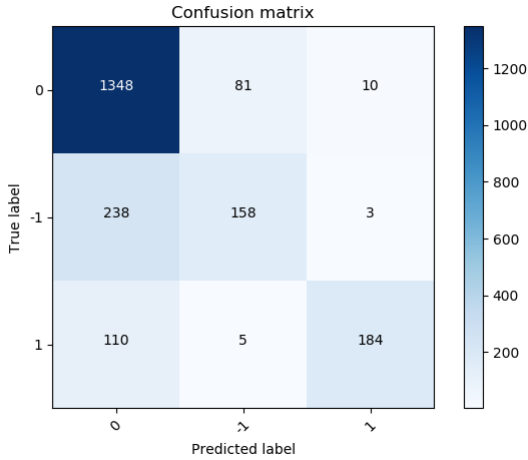
- Skenario *Last*



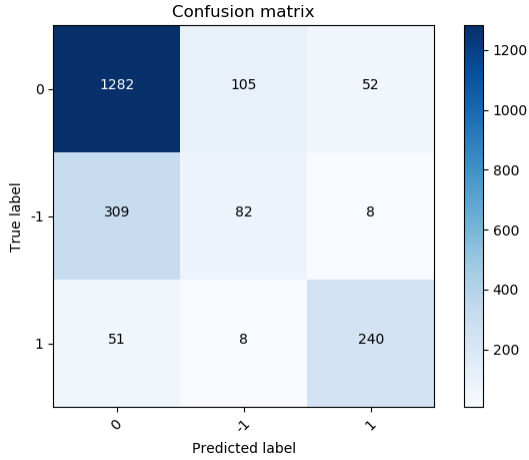
- Skenario *Freez*



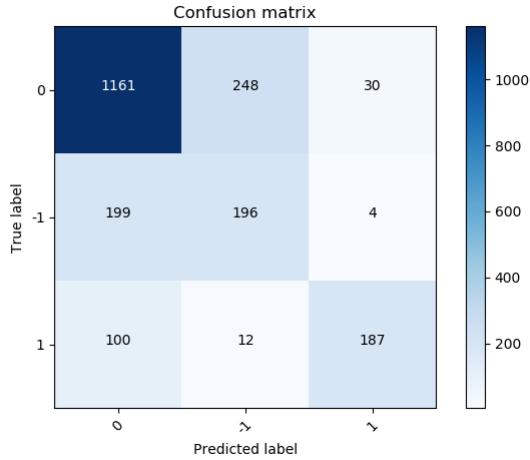
- Skenario *Freez + Discr*



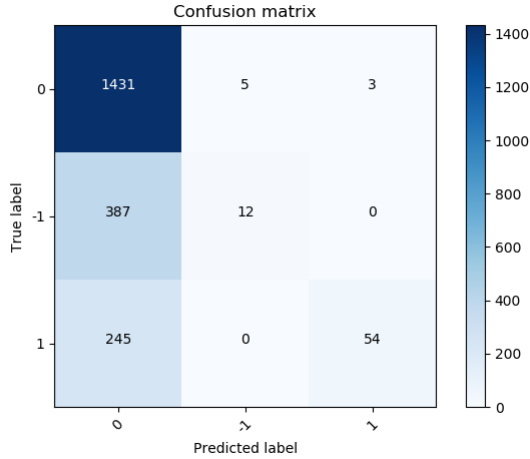
- Skenario *Freez + STLR*



- Skenario *Freez + Cos*



- Skenario *Freez + Discr + STLR*



LAMPIRAN H: LOSS & AKURASI CLASSIFIER SUBTASK C

- Skenario *Full*

epoch	train_loss	valid_loss	accuracy
1	0.860552	0.697372	0.770239
2	0.554967	0.60758	0.777258
3	0.408232	0.638838	0.773982
4	0.263667	0.688931	0.766963
5	0.188442	0.878301	0.768367
6	0.149958	0.929336	0.774918
7	0.099955	0.911755	0.776322
8	0.109552	0.987702	0.783341
9	0.075813	1.013452	0.766027
10	0.081551	1.025363	0.769303
11	0.059877	1.00752	0.775854
12	0.077635	1.042033	0.768835
13	0.055196	1.028846	0.769771
14	0.096476	1.003692	0.759008
15	0.069649	1.025519	0.751989

- Skenario *Full + Discr*

epoch	train_loss	valid_loss	accuracy
1	0.657118	0.601072	0.776322
2	0.521696	0.617788	0.781001
3	0.350256	0.685376	0.763687
4	0.229927	0.944785	0.776322
5	0.132278	1.092833	0.742162
6	0.109683	1.153185	0.761348
7	0.103222	1.137679	0.729995
8	0.097917	1.111575	0.773046

9	0.078363	1.123798	0.757136
10	0.090603	1.363946	0.777726
11	0.081432	1.426685	0.773046
12	0.071251	1.441039	0.77679
13	0.061461	1.30005	0.760412
14	0.078253	1.145447	0.748245
15	0.055603	1.215749	0.781937

- Skenario *Last*

epoch	train_loss	valid_loss	accuracy
1	0.893549	0.757523	0.74263
2	0.686727	0.711061	0.761348
3	0.622181	0.713906	0.758072
4	0.601015	0.728581	0.759476
5	0.56807	0.692611	0.762752
6	0.585852	0.776742	0.749649
7	0.540553	0.747298	0.757136
8	0.529257	0.752589	0.759944
9	0.504573	0.945044	0.715957
10	0.498982	0.750247	0.759008
11	0.475665	0.816149	0.757136
12	0.476517	0.748107	0.761816
13	0.446928	0.791947	0.763687
14	0.447614	0.786528	0.749181
15	0.439444	0.809094	0.757136

- Skenario *Freez*

epoch	train_loss	valid_loss	accuracy
1	0.893549	0.757523	0.74263

epoch	train_loss	valid_loss	accuracy
1	0.674912	0.686141	0.763687

epoch	train_loss	valid_loss	accuracy
1	0.574518	0.620987	0.769771

epoch	train_loss	valid_loss	accuracy
1	0.450782	0.609218	0.77679
2	0.282039	0.728486	0.766027
3	0.202246	0.876631	0.754796
4	0.129239	0.867455	0.762752
5	0.102227	0.911608	0.766495
6	0.083617	0.985248	0.755732
7	0.086027	0.992007	0.744034
8	0.070276	0.994655	0.766495
9	0.080007	1.025586	0.771175
10	0.05724	0.966529	0.763219
11	0.078492	1.069976	0.765091
12	0.071424	1.123694	0.768367
13	0.069509	1.116352	0.717361
14	0.059645	1.064004	0.775386
15	0.045805	1.14527	0.771175

- Skenario *Freez + Discr*

epoch	train_loss	valid_loss	accuracy
1	0.718008	0.693883	0.745905

epoch	train_loss	valid_loss	accuracy
1	0.652529	0.618027	0.772578

epoch	train_loss	valid_loss	accuracy
1	0.548186	0.649557	0.773982

epoch	train_loss	valid_loss	accuracy
1	0.373897	0.733615	0.780533
2	0.225698	0.852759	0.769303
3	0.167457	1.145819	0.773046
4	0.119338	1.034464	0.751053
5	0.110477	1.100255	0.766027
6	0.093176	1.121927	0.735143
7	0.100727	1.374312	0.777258
8	0.06841	1.321485	0.77211
9	0.078591	1.301281	0.770239
10	0.062015	1.290749	0.771643
11	0.113749	1.131963	0.755732
12	0.082314	1.344624	0.761816
13	0.077847	1.329245	0.708002
14	0.069411	1.26593	0.771643
15	0.057589	1.413165	0.779598

- Skenario *Freez + STLR*

epoch	train_loss	valid_loss	accuracy
1	1.088567	0.962795	0.731867

epoch	train_loss	valid_loss	accuracy

1	1.116276	1.001406	0.711277

epoch	train_loss	valid_loss	accuracy
1	1.001078	0.983466	0.72204

epoch	train_loss	valid_loss	accuracy
1	0.918641	0.994009	0.709406
2	1.035978	1.020963	0.693028
3	1.084904	1.052351	0.664951
4	1.114609	1.102786	0.636874
5	1.141211	1.140593	0.615817
6	1.159902	1.163098	0.607394
7	1.179063	1.183656	0.597567
8	1.170454	1.21125	0.58306
9	1.163371	1.247779	0.567618
10	1.170343	1.277156	0.553112
11	1.191929	1.279881	0.556387
12	1.186882	1.31233	0.543285
13	1.174401	1.329344	0.536734
14	1.177157	1.345328	0.530182
15	1.170093	1.363281	0.522695

- Skenario *Freez + Cos*

epoch	train_loss	valid_loss	accuracy
1	0.880606	0.748839	0.736079
2	0.688081	0.713376	0.756668
3	0.618205	0.740515	0.745438
4	0.600255	0.713373	0.755732
5	0.561675	0.687128	0.761816

6	0.585385	0.763592	0.751521
7	0.533398	0.802951	0.725316
8	0.523198	0.771027	0.743098
9	0.502537	0.860409	0.714085
10	0.492646	0.815504	0.731399
11	0.481438	0.837744	0.732335
12	0.476337	0.721037	0.761348
13	0.457434	0.719129	0.765091
14	0.451274	0.796044	0.728124
15	0.432568	0.768935	0.753861

epoch	train_loss	valid_loss	accuracy
1	0.323407	0.839932	0.729527
2	0.44065	0.813379	0.730463
3	0.445326	0.796674	0.733739
4	0.527514	0.787803	0.733739
5	0.566689	0.78003	0.736079
6	0.558173	0.778692	0.730931
7	0.557652	0.772605	0.729995
8	0.537868	0.765955	0.728592
9	0.541971	0.768599	0.723912
10	0.525171	0.771231	0.723444
11	0.534003	0.772049	0.720168
12	0.519266	0.774719	0.714085
13	0.509352	0.781473	0.711745
14	0.505574	0.791149	0.707066
15	0.486713	0.791582	0.705194

epoch	train_loss	valid_loss	accuracy
1	0.433002	0.805619	0.703322

2	0.423107	0.775763	0.720168
3	0.439372	0.759137	0.723912
4	0.421466	0.744394	0.731867
5	0.396875	0.737042	0.733271
6	0.379573	0.735672	0.733271
7	0.363754	0.73248	0.734207
8	0.3635	0.734727	0.729059
9	0.353055	0.73715	0.72672
10	0.34994	0.741274	0.726252
11	0.338931	0.741397	0.726252
12	0.339033	0.764556	0.710342
13	0.331084	0.76738	0.714553
14	0.320794	0.759524	0.717829
15	0.318688	0.767198	0.716425

epoch	train_loss	valid_loss	accuracy
1	0.430446	0.811211	0.716893
2	0.379421	0.821496	0.70847
3	0.366544	0.779553	0.728592
4	0.329441	0.812383	0.707534
5	0.315333	0.813039	0.717829
6	0.328859	0.809142	0.728124
7	0.308211	0.792857	0.731867
8	0.292016	0.783072	0.736079
9	0.283009	0.779288	0.736079
10	0.270646	0.776053	0.738886
11	0.259167	0.776002	0.738886
12	0.24779	0.77468	0.728592
13	0.247832	0.785742	0.726252
14	0.247205	0.782432	0.720636

15	0.24575	0.780545	0.720168

- Skenario *Freez + Discr + STLR*

epoch	train_loss	valid_loss	accuracy
1	0.72725	0.76163	0.699579

epoch	train_loss	valid_loss	accuracy
1	0.755123	0.763169	0.697239

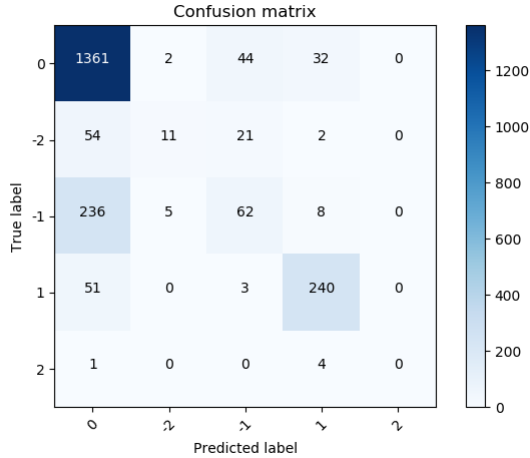
epoch	train_loss	valid_loss	accuracy
1	0.840467	0.759736	0.699579

epoch	train_loss	valid_loss	accuracy
1	0.729243	0.766965	0.693964
2	0.96973	0.757049	0.698643
3	1.02092	0.748913	0.705194
4	1.066294	0.740309	0.713617
5	1.083613	0.734996	0.720168
6	1.140506	0.730586	0.724848
7	1.154873	0.727437	0.729995
8	1.150469	0.725442	0.736547
9	1.148587	0.722117	0.745905
10	1.157262	0.719982	0.749181
11	1.191948	0.719855	0.748713
12	1.188579	0.720304	0.750117
13	1.158284	0.720039	0.751521

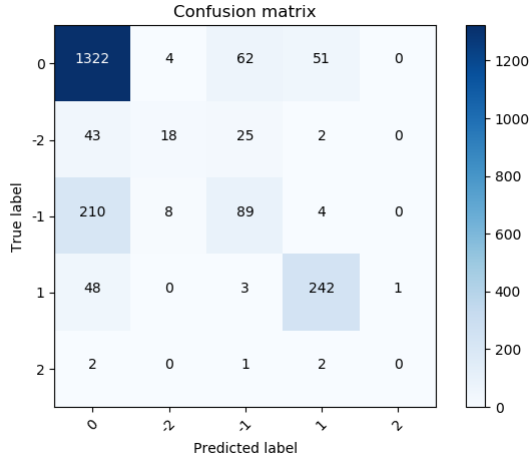
14	1.147592	0.720841	0.749181
15	1.130436	0.719989	0.751053

LAMPIRAN I: CONFUSION MATRIX SUBTASK C

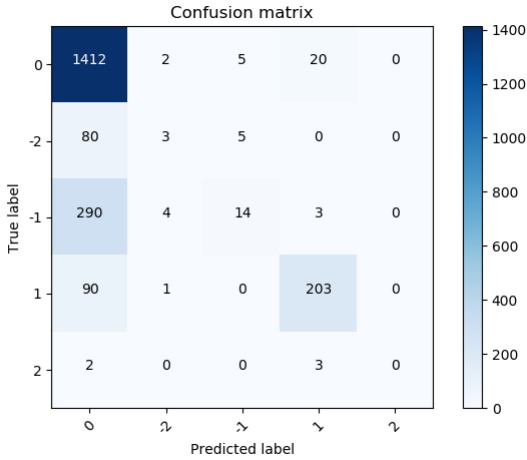
- Skenario *Full*



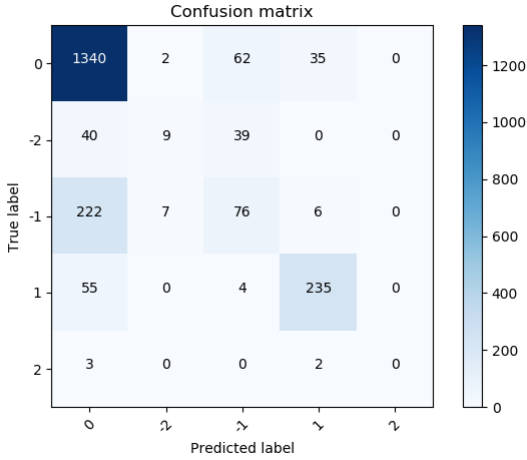
- Skenario *Full + Discr*



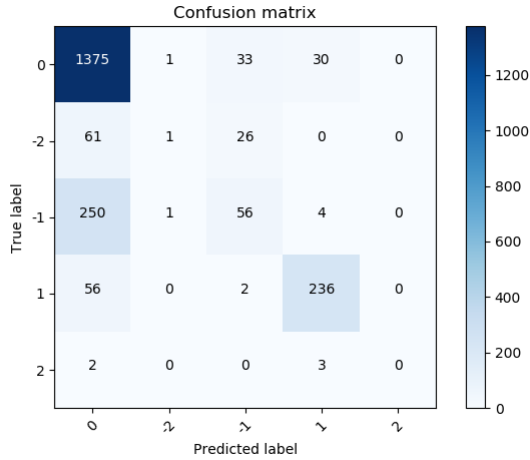
- Skenario *Last*



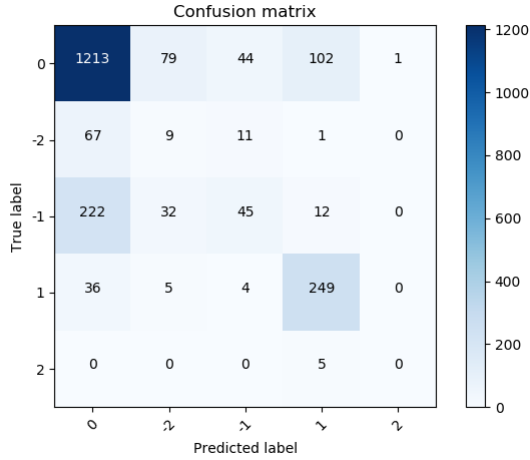
- Skenario *Freez*



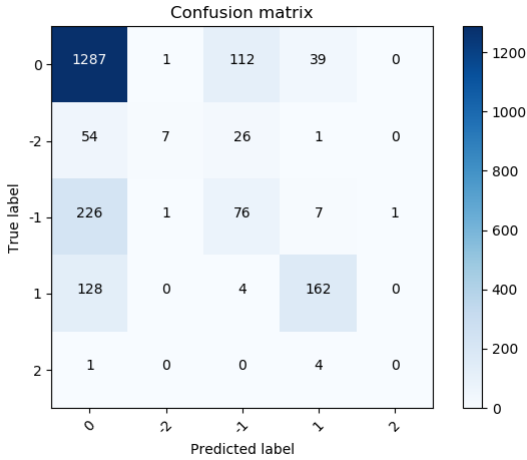
- Skenario *Freez + Discr*



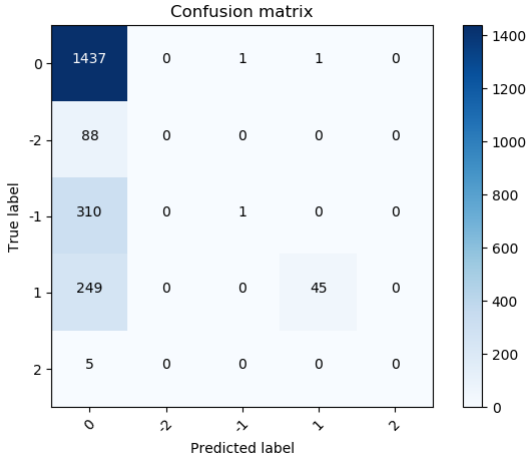
- Skenario *Freez + STLR*



- Skenario *Freez + Cos*



- Skenario *Freez + Discr + STLR*



LAMPIRAN J : F1, AKURASI, PRESISI, DAN RECALL SELURUH SUBTASK

- **Subtask A**

Skenario	f1_micro	f1_macro	f1_weighted	precision_micro	precision_macro	precision_weighted	recall_micro	recall_macro	recall_weighted	accuracy
Full	0.958	0.957	0.958	0.958	0.957	0.958	0.958	0.957	0.958	0.958
Full + discr	0.950	0.949	0.950	0.950	0.950	0.950	0.950	0.949	0.950	0.950
Last	0.929	0.927	0.929	0.929	0.931	0.930	0.929	0.925	0.929	0.929
Freez	0.946	0.945	0.946	0.946	0.947	0.946	0.946	0.944	0.946	0.946
Freez + discr	0.946	0.945	0.946	0.946	0.949	0.947	0.946	0.942	0.946	0.946
Freez + stlr	0.925	0.923	0.925	0.925	0.928	0.926	0.925	0.920	0.925	0.925
Freez + cos	0.924	0.923	0.924	0.924	0.921	0.926	0.924	0.927	0.924	0.924
Freez + discr + stlr	0.918	0.914	0.917	0.918	0.932	0.925	0.918	0.907	0.918	0.918

- **Subtask B**

Skenario	f1_micro	f1_macro	f1_weighted	precision_micro	precision_macro	precision_weighted	recall_micro	recall_macro	recall_weighted	accuracy
Full	0.806	0.712	0.784	0.806	0.807	0.799	0.806	0.675	0.806	0.806
Full + discr	0.809	0.720	0.789	0.809	0.809	0.802	0.809	0.682	0.809	0.809
Last	0.786	0.645	0.741	0.786	0.800	0.784	0.786	0.630	0.786	0.786
Freez	0.807	0.713	0.785	0.807	0.799	0.798	0.807	0.679	0.807	0.807
Freez + discr	0.791	0.698	0.775	0.791	0.792	0.787	0.791	0.649	0.791	0.791
Freez + stlr	0.751	0.637	0.724	0.751	0.667	0.716	0.751	0.633	0.751	0.751
Freez + cos	0.723	0.660	0.726	0.723	0.690	0.734	0.723	0.641	0.723	0.723
Freez + discr + stlr	0.701	0.393	0.604	0.701	0.782	0.731	0.701	0.402	0.701	0.701

- **Subtask C**

Skenario	f1_micro	f1_macro	f1_weighted	precision_micro	precision_macro	precision_weighted	recall_micro	recall_macro	recall_weighted	accuracy
Full	0.783	0.437	0.747	0.783	0.545	0.748	0.783	0.417	0.783	0.783
Full + discr	0.782	0.469	0.758	0.782	0.542	0.755	0.782	0.447	0.782	0.782
Last	0.764	0.355	0.695	0.764	0.505	0.727	0.764	0.350	0.764	0.764
Freez	0.777	0.433	0.747	0.777	0.514	0.742	0.777	0.415	0.777	0.777
Freez + discr	0.781	0.396	0.735	0.781	0.493	0.733	0.781	0.390	0.781	0.781
Freez + stlr	0.709	0.373	0.687	0.709	0.394	0.690	0.709	0.387	0.709	0.709
Freez + cos	0.717	0.378	0.689	0.717	0.529	0.698	0.717	0.354	0.717	0.717
Freez + discr + stlr	0.694	0.217	0.586	0.694	0.433	0.671	0.694	0.231	0.694	0.694

Halaman ini sengaja dikosongkan.

BIODATA PENULIS

Penulis lahir di Nganjuk pada tanggal 28 Juni 1997. Merupakan anak pertama dari 2 bersaudara. Penulis telah menempuh beberapa pendidikan formal yaitu; SD Negeri 3 Nglawak, SMP Negeri 1 Kertosono, dan SMA Negeri 1 Kertosono.

Pada Tahin 2015 pasca kelulusan SMA, penulis melanjutkan pendidikan di Jurusan Sistem Informasi Fakultas Teknologi Informasi dan Komunikasi – Institut Teknologi Sepuluh Nopember (ITS) Surabaya dan terdaftar sebagai mahasiswa dengan NRP 052115410000051. Selama menjadi mahasiswa, penulis mengikuti berbagai kegiatan kemahasiswaan seperti beberapa kepanitian serta pernah menjabat sebagai Kepala Departemen Ristek BEM FTIK ITS. Di bidang akademik, penulis aktif menjadi asisten dosen praktikum pada mata kuliah Sistem Enterprise, Pengantar Sistem Operasi, dan Pemrograman Berbasis Web.

Pada tahun keempat, karena penulis memiliki ketertarikan di bidang pengolahan data, maka penulis mengambil bidang minat Akuisisi Data dan Diseminasi Informasi (ADDI). Penulis dapat dihubungi melalui *email* di andiragita28@gmail.com.