DETEKSI EMOSI MENGGUNAKAN CITRA EKSPRESI WAJAH SECARA OTOMATIS

Disusun Oleh:

N a m a	: Faza Nur Azizi
NIM	: 17523112

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2021

HALAMAN PENGESAHAN DOSEN PEMBIMBING

DETEKSI EMOSI MENGGUNAKAN EKSPRESI WAJAH SECARA OTOMATIS

TUGAS AKHIR

Disusun Oleh:

N a m a : Faza Nur Azizi

NIM : 17523112

Yogyakarta, 27 Agustus 2021

Pembimbing,

(Arrie Kurniawardhani, S.Si., M.Kom.)

HALAMAN PENGESAHAN DOSEN PENGUJI

DETEKSI EMOSI MENGGUNAKAN EKSPRESI WAJAH SECARA OTOMATIS

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 26 Agustus 2021

Tim Penguji

Arrie Kurniawardhani, S.Si., M.Kom. _____

Anggota 1

Sri Mulyati, S.Kom., M.Kom. _____

Anggota 2

Andhika Giri Persada, S.Kom., M.Eng. _____

Mengetahui,
Ketua Program Studi Informatika – Program Sarjana
Fakultas Teknologi Industri
Universitas Islam Indonesia

(Dr. Raden Teduh Dirgahayu, S.T., M.Sc.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Faza Nur Azizi

NIM : 17523112

Tugas akhir dengan judul:

DETEKSI EMOSI MENGGUNAKAN EKSPRESI WAJAH SECARA OTOMATIS

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 31 Agustus 2021

(Faza Nur Azizi)

HALAMAN PERSEMBAHAN

Alhamdulilah kupanjatkan kepada Allah SWT, atas segala rahmat dan juga kesempatan dalam menyelesaikan tugas akhir saya yang berjudul "Deteksi Emosi Menggunakan Ekspresi Wajah Secara Otomatis" dengan segala kekurangannya. Dan Segala puji dan syukur bagi Allah SWT, karena telah menghadirkan orang-orang berarti disekeliling saya yang selalu memberi semangat dan doa sehingga skripsi saya ini dapat diselesaikan dengan baik. Tugas akhir ini saya persembahkan untuk:

1. Bapak Imam Effendy M.A dan Ibu Wiwik Wijayati Amaliyah
2. Keempat kakak laki-laki saya Fanni Arifian, Ferdian Wildana, Frisky Adi Sakti, dan Fikri Amali
3. Saudara kembar saya Fifi Nur Azizati
4. Rekan seperjuangan saya di jurusan Informatika FTI UII
5. Teman-teman squad "DQ" Asyam Mahardika, Faiz Irsyad, Rafael Jody, dan Sheikal Ilyasa
6. Ibu Arrie Kurniawardhani, S.Si., M.Kom. selaku dosen pembimbing
7. Jurusan Informatika Universitas Islam Indonesia yang telah mewadahi saya dalam menuntut ilmu dan sebagai wadah untuk saya mengembangkan diri.
8. Serta seluruh pihak yang tidak bisa saya sebutkan satu-persatu, terimakasih ata segala perhatian dan dukungan yang luar biasa.

HALAMAN MOTO

“The Way Get Started Is To Quit Talking And Begin Doing.”

(Walt Disney)

“Don’t Let Yesterday Take Up Too Much Of Today.”

(Will Rogers)

"Sesungguhnya sesudah kesulitan itu ada kemudahan, sesungguhnya sesudah kesulitan itu ada kemudahan."

(Al-Insyirah ayat 5-6)

"Allah tidak membebani seseorang melainkan sesuai kesanggupannya."

(Al Baqarah ayat 286)

KATA PENGANTAR

Assalamualaikum Warahmatullahi Wabarakatuh.

Puji dan syukur penulis panjatkan kehadirat Allah SWT, karena berkat rahmat dan karunia-Nyalah penulis dapat menyelesaikan laporan tugas akhir yang berjudul "Deteksi Emosi Menggunakan Citra Ekspresi Wajah Secara Otomatis". Adapun maksud dan tujuan dari penulisan skripsi ini adalah untuk memenuhi salah satu syarat untuk mengikuti sidang skripsi, Jurusan Informatika Fakultas Teknik Industri Universitas Islam Indonesia. Penulis sadar, tanpa dukungan serta motivasi laporan ini tidak akan dapat selesai tepat pada waktunya. Oleh karena itu, penulis menyampaikan rasa terimakasi yang sebesar-besarnya kepada:

1. Bapak Fathul Wahid, S.T., M.Sc., Ph.D., selaku selaku Rektor Universitas Islam Indonesia.
2. Bapak Prof. Dr. Ir. Hari Purnomo selaku Dekan Fakultas Teknologi Industri Universitas Islam Indonesia.
3. Bapak Hendrik, S.T., M.Eng., selaku Ketua Jurusan Informatika Fakultas Teknologi Industri Universitas Islam Indonesia.
4. Ibu Arrie Kurniawardhani, S.Si., M.Kom., selaku dosen pembimbing, untuk semua nasihat, bimbingan, ilmu, dan dukungan yang luar biasa yang selama ini telah diberikan.
5. Bapak Imam Effendy M.A dan Ibu Wiwik Wijayati Amaliyah untuk doa dan dukungan nya untuk menyelesaikan studi ini.
6. Keempat kakak laki-laki saya Fanni Arifian, Ferdian Wildana, Frisky Adi Sakti, dan Fikri Amali atas dukungan dan doa nya.
7. Saudara kembar saya Fifi Nur Azizati atas semangat dan dukungannya
8. Sahabat-sahabat saya Prizega Fromadia Godradiansyah, Asyam Mahardika Putra, Faiz Irsyad Kuncoro, Rafael Jody Alvian, dan Sheikal Ilyasa Kirana yang selalu menemani dan memberi semangat.
9. Teman-teman PIXEL/Informatika angkatan 2017.
10. Semua pihak yang tidak bisa saya sebutkan satu per satu, terimakasih atas semua bentuk dukungannya.

Penulis menyadari bahwa penyusunan skripsi ini jauh dari sempurna, semoga Allah SWT memberikan balasan yang berlipat ganda kepada semua pihak yang telah turut membantu

penulis dalam menyelesaikan penulisan skripsi ini. Oleh karena itu, penulis berharap atas saran dan kritik yang bersifat membangun dari pembaca.

Akhir kata, penulis mengharapkan semoga tujuan dari pembuatan skripsi ini dapat tercapai sesuai dengan yang diharapkan.

Yogyakarta, 14 Juni 2021

Faza Nur Azizi

SARI

Ekspresi wajah merupakan metode yang paling efektif bagi manusia untuk mengekspresikan emosi. Ekspresi wajah memiliki keunggulan untuk mengetahui emosi seseorang, karena ketika seorang secara emosi tidak stabil akan nampak pada raut wajah yang berubah, seperti kerutan pada kening, kedipan pada mata, ataupun perubahan warna pada kulit wajah. CNN merupakan pengembangan dari MLP yang dirancang untuk mengolah data dua dimensi. Dua tahapan utama pada CNN yaitu *feature learning* dan *classification*. CNN merupakan metode yang digunakan dalam penelitian. Penelitian ini menggunakan dataset dari FER2013, KDEF, dan CK+. Dataset-dataset tersebut berisikan citra greyscale dengan format jpeg. Arsitektur model yang dibuat memiliki empat lapisan konvolusi dan dua lapisan *fully connected.* Penelitian dilakukan dengan mencoba tiga optimasi yaitu, ADAM, SGD, dan RMS. Kemudian juga dilakukan percobaan dengan perbedaan *epoch.* Hasil penelitian ini menunjukkan optimasi ADAM memiliki nilai akurasi paling tinggi dibandingkan dengan optimasi lainnya. Akurasi tertinggi pada dataset KDEF dengan jumlah 82%. Emosi yang mudah dikenali atau yang memiliki akurasi tinggi yaitu jijik dan emosi yang susah dikenali yaitu sedih. Dapat disimpulkan CNN dapat digunakan untuk deteksi emosi melalui citra ekspresi dengan akurasi yang baik.

Kata kunci : ekspresi, deteksi emosi, *Convolutional Neural Network.*

GLOSARIUM

CNN	Salah satu jenis neural network yang biasa digunakan pada data image
Data	Merupakan sekumpulan keterangan atau fakta yang dibuat dengan simbol, angka, kata-kata, maupun kalimat
Flowchart	Sebuah bagian dengan simbol (sandi) tertentu yang menjelaskan dan menggambarkan langkah-langkah proses secara mendetail, dan hubungan antara proses (metode) dengan proses lainnya pada suatu program
JPEG	Ekstensi file gambar yang pertama kali muncul, merupakan kepanjangan dari *Joint Photographic Experts Group* yang mulai diperkenalkan pada tahun 1992 sebagai standar ISO
Python	Bahasa pemrograman interpretatif multiguna

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Ekspresi wajah merupakan metode yang paling efektif bagi manusia untuk mengekspresikan emosi. Satu emosi dapat memberi lebih banyak informasi daripada kata-kata (Oliver & Alcover, 2020). Ekspresi wajah memiliki keunggulan untuk mengetahui emosi seseorang, karena ketika seorang secara emosi tidak stabil akan nampak pada raut wajah yang berubah, seperti kerutan pada kening, kedipan pada mata, ataupun perubahan warna pada kulit wajah (Planalp, 2015). Biasanya ketika perasaan seseorang berubah, ekspresi wajahnya juga akan berubah. Misalnya ketika merasa marah, kening dan wajah akan menunjukkan ciri-ciri seperti mengerutkan kening dan wajah menjadi merah. Perasaan yang mendorong individu untuk bertindak atau merespon terhadap stimulus disebut emosi (Amynarto et al., 2018).

Pada tahun 2019 di jepang, Toyota membuat sebuah mobil yang dapat mendeteksi emosi seseorang melalui kamera yang terdapat pada *dashboard* mobil. Sebagai contoh, ketika sedang merasa sedih ketika berada di dalam mobil. Maka akan dilakukan pendeteksian emosi dari pengendara kemudian dianalisa oleh *Artificial Intelligence* yang tertanam pada mobil tersebut dan menghasilkan rekomendasi-rekomendasi yang menjadi solusi bagi pengendara. Tak hanya itu pendeteksian emosi penting dilakukan pada aspek psikologi, ketika dalam melakukan konseling seorang konselor melakukan komunikasi dengan pasien. Komunikasi bisa terjadi dengan komunikasi verbal dan komunikasi non verbal. Komunikasi verbal yaitu komunikasi yang menggunakan kata-kata, entah lisan maupun tulisan dan komunikasi non verbal yaitu komunikasi yang pesannya dikemas dalam bentuk tanpa kata-kata (Kusumawati, 2016). Ketika psikolog mengetahui emosi yang dialami pasien maka psikolog dapat mengambil langkah yang tepat untuk pasien.

Penelitian mengenai deteksi emosi sudah dilakukan oleh beberapa peneliti. *Convolutional Neural Network* (CNN) merupakan metode yang memiliki hasil yang sginifikan. CNN berusaha meniru visual cortex manusia dalam pengenalan citra sehingga memiliki kemampuan mengolah informasi citra(Eka Putra, 2016). Oleh karena itu, penelitian ini menggunakan metode CNN untuk melakukan deteksi emosi yang memiliki input data citra ekspresi wajah.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dijelaskan sebelumnya, maka dapat dirumuskan yaitu bagaimana cara deteksi emosi dengan menggunakan citra ekspresi wajah?

1.3 Batasan Masalah

Penelitian ini memiliki beberapa batasan agar tidak adanya penyimpangan pada pembahasan. Berikut batasan-batasan masalah penelitian ini:

a. Dataset yang digunakan pada penelitian ini yaitu, Facial Expression Recognition 2013(FER2013), Cohn-Kanade Dataset (CK+), dan Karolinska Directed Emotional Faces (KDEF).
b. Terdapat tujuh emosi yang digunakan. Emosi tersebut adalah marah, jijik, takut, senang, sedih, terkejut, dan netral.
c. Citra yang digunakan adalah citra ekspresi wajah seseorang dan berwarna *grayscale*.
d. Dataset memiliki ukuran 48x48 *pixels*.
e. Format citra yaitu JPEG.

1.4 Tujuan Penelitian

Penelitian ini memiliki tujuan untuk mengetahui emosi seseorang melalui pembacaan ekspresi wajah pada gambar dengan menggunakan metode *Convolution Neural Network*.

1.5 Manafaat Penelitian

Harapan dengan terselesainya penelitian ini akan membawa beberapa manfaat untuk aspek-aspek tertentu. Manfaat penelitian ini yaitu, penelitian dapat ini menjadi tempat untuk menambah ilmu mengenai *deep learning* khusus nya pada *convolutional neural network* untuk pemrosesan citra. Dari penelitian ini dapat diketahui teori hingga penerapan metode tersebut.

1.6 Metodologi Penelitian

Langkah-langkah yang digunakan pada penelitian agar tercapainya tujuan penelitian adalah sebagai berikut:

a. Pengumpulan Data

Tahap ini akan menghasilkan data yang akan digunakan dalam penelitian. Data ini merupakan citra emosi dan akan dijadikan sebagai data latih, data uji, dan data validasi.

b. Perancangan Sistem

Tahap ini akan dilakukan penggambaran dari perencanaan sistem sehingga penelitian dapat terstruktur dan dapat memudahkan peneliti.

1.7 Sistematika Laporan

Dalam penyusunan laporan tugas akhir ini, sistematika penulisan dibagi menjadi beberapa bab sebagai berikut:

Bab I Pendahuluan

Bab ini berisi tentang latar belakang permasalahan dari penelitian deteksi emosi menggunakan citra. Dari latar belakang tersebut kemudian dibuat rumusan masalah, tujuan penelitian, manfaat penelitian, dan sistematika penelitian.

Bab II Landasan Teori

Bab ini berisi tentang teori-teori dan konsep yang berhubungan dengan penelitian. Dengan adanya teori dan konsep digunakan untuk mendukung pemecahan permasalahan. Teori-teori dan konsep didapat dari sumber-sumber seperti, jurnal, buku, dan artikel.

Bab III Metodologi Penelitian

Bab ini berisi tahapan-tahapan yang dilakukan pada penelitian. Tahapan-tahapan tersebut yaitu pengumpulan data, membagi data, dan tahapan yang ada pada metode CNN seperti membangun arsitektur model, melatih model, dan melakukan pengujian.

Bab IV Hasil dan Pembahasan

Bab ini berisi proses penelitian yang dijelaskan juga kode-kode yang digunakan. Dan bab terdapat hasil dari pengujian dari model yang telah dibangun.

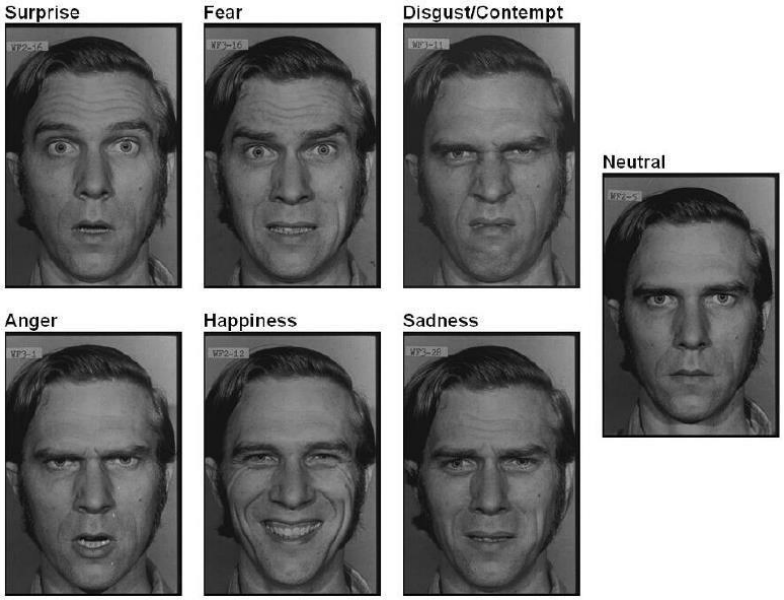
Bab V Kesimpulan dan Saran

Bab ini berisi kesimpulan yang diperoleh dari hasil penelitian dan analisa data yang telah dilakukan dan saran-saran yang dapat diterapkan dari hasil yang dapat menjadi masukan yang berguna kedepannya

BAB II
LANDASAN TEORI

2.1 Emosi

Emosi adalah sebuah perasaan yang dapat mendorong seseorang untuk bertindak ataupun merespon dari suatu stimulus (Goleman, 2002). Emosi seseorang dapat diketahui dengan melihat dan mengamati ekspresi mikro seseorang tersebut, ekspresi mikro yang ditunjukkan oleh seseorang merupakan hal yang bersifat universal bagi seluruh manusia (Ekman, 2016). Ekspresi mikro merupakan gerakan wajah singkat yang mencoba untuk menyembunyikan emosi saat sedang mengungkapkan emosi yang dialami(Amynarto et al., 2018). Emosi dapat dilihat dari perubahan pada raut wajah, seperti kerutan pada kening dan kedipan mata(L.Pt. Purnamaningsih, Ni Kt. Suarni, 2019).



Gambar 2.1 Contoh citra emosi

Sumber: (Fabri, 2004)

2.2 Citra

Citra merupakan gambar dua dimensi yang dapat dihasilkan melalui proses sampling dari gambar analog yang kontinu menjadi gambar diskrit. Proses sampling tersebut terbagi menjadi dua, yaitu *downsampling* dan *upsampling*. *Downsampling* adalah proses yang akan menghasilkan nilai citra lebih kecil dengan menurunkan jumlah piksel atau resolusi citra

spasial. Sedangkan *upsampling* merupakan kebalikan dengan *downsampling*, yaitu proses yang dapat menaikkan resolusi gambar atau jumlah piksel gambar(Nabuasa, 2019).

2.3 Deteksi Wajah

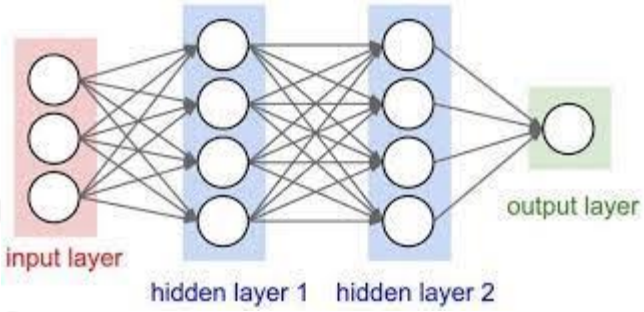
Sistem pengenalan wajah memiliki konsep yang hampir sama dengan sistem pengenalan *biometric* lainnya seperti *palm recognition* dan *fingerprint*. Bahwa pada setiap individu memiliki fitur dan beberapa struktur wajah yang unik atau berbeda satu dengan yang lain. Oleh karena itu, pengenalan wajah secara otomatis dapat dilakukan dengan pembacaan simetris dari wajah masing-masing individu. Sistem pengenalan wajah mempunyai kesulitan tersendiri, antara lain: (1) Titik skala dan pergeseran pada wajah (2) Perbedaan pada paras wajah, seperti pose wajah, segi wajah, model rambut, makeup, jenggot, kumis, aksesoris yang dikenakan, dan lain-lain (3) Pencahayaan (4) Usia (Çarıkçı & Özen, 2012). Maka hampir semua area wajah menjadi hal penting dalam mendeteksi emosi seseorang, terutama pada bagian wajah dan bibir sebagai area yang dapat menggambarkan emosi seseorang dengan jelas(Tri Anindia Putra, 2015).

Pengenalan wajah secara umum terbagi menjadi dua tahapan yaitu, mendeteksi wajah (pre-processing) untuk mengekstraksi ciri dari wajah dan sistem pengenalan wajah (face recognition) (Alexander, 2013). Metode pengenalan wajah pada seseorang dapat menjadi sebuah input yang dapat membantu dalam mendeteksi ekspresi wajah seseorang. Ekspresi maupun emosi individu menjadi salah satu cara untuk berkomunikasi secara tidak langsung atau komunikasi nonverbal. Karena secara tidak langsung ekspresi wajah yang terdiri dari pergerakan otot wajah dapat mengungkapkan isi hati kepada seseorang yang saling berkomunikasi. Contoh dalam berekspresi antara lain, mengernyitkan alis mata dapat menunjukan kemarahan atau ketidaksukaan, mengangkat alis dapat menunjukan ekspresi heran dan terkejut. Semua emosi dan berbagai macam isi hati manusia tergambar pada ekspresi wajah yang berbeda-beda(Abidin, 2011).

2.4 *Convolutional Neural Network* (CNN)

CNN merupakan sebuah metode pengembangan dari *multilayer perceptron*(MLP) yang digunakan untuk megolah data dua dimensi. Salah satu yang termasuk *deep neural network* yaitu CNN, dikarenakan memiliki kedalaman jaringan yang tinggi dan banyak diaplikasikan untuk citra(Eka Putra, 2016). Dua tahapan utama pada CNN yaitu *feature learning* dan *classification*. Tahapan *feature learning* terdiri dari *convolution layer*, *ReLU* (fungsi aktivasi)

dan *pooling layer* sedangkan pada tahap *classification* terdiri dari *flatten*, *fully-connected layer*, dan prediksi. Cara kerja hampir mirip dengan MLP yang membedakan CNN dipresentasikan dalam bentuk dua dimensi(Achmad et al., 2019).



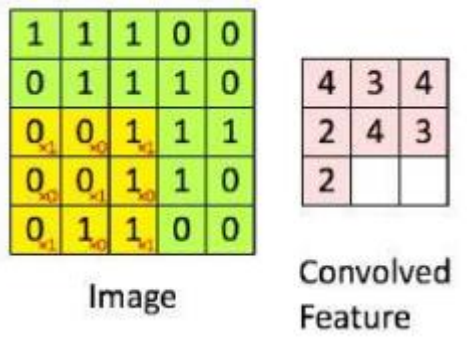
Gambar 2.2 Arsitektur MLP

Sumber: (Eka Putra, 2016)

Gambar 2.2 menjelaskan terkait konsep kerja dari arsitektur MLP. Kotak merah dan biru merupakan sebuah *layer* yang berisikan neuron yang digambarkan lingkaran putih. *Input* dari MLP merupakah sebuah data satu dimensi yang kemudian data tersebut di propagasikan pada jaringan hingga mendapatkan *output*. Hubungan satu neuron dengan neuron lain yang terdapat pada dua layer yang bersebelahan memiliki parameter bobot. Data *input* akan dilakukan operasi linier dengan bobot yang sudah ada pada setiap layer dan hasil komputasi ditransformasi menggunakan operasi linear yaitu fungsi aktivasi. Sedangkan pada CNN data yang akan di propagasikan merupakan data dua dimensi yang membuat operasi linear dan parameter bobot berbeda dengan MLP. Operasi linier yang digunakan CNN yaitu operasi konvolusi dan bobot tidak lagi satu dimensi(Eka Putra, 2016).

2.4.1 Konvolusi

Konvolusi merupakan proses utama yang menjadi dasar dalam sebuah CNN. Operasi konvolusi pada output dari layer sebelumnya disebut *Convolution layer*. Hal itu dapat diartikan sebagai mengaplikasikan sebuah fungsi pada *output* fungsi lain secara berulang. Konvolusi pada data citra dilakukan dengan pengaplikasian *kernel* ke *offset* yang ada pada citra. Dapat dilihat pada Gambar 2.3.



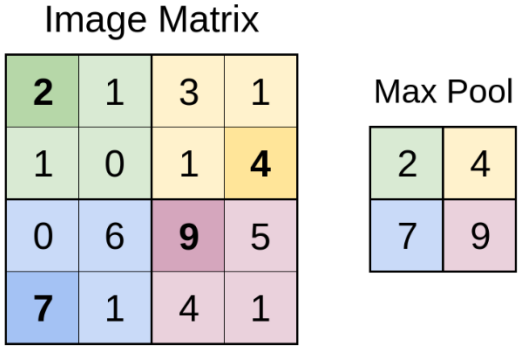
Gambar 2.3 Operasi konvolusi

Sumber: (Eka Putra, 2016)

Pada Gambar 2.3 kotak kuning merupakan *kernel* dan kotak hijau *offset* dari data citra. *Kernel* bergerak dari kiri atas menuju kanan bawah. Konvolusi bertujuan untuk mengekstraksi fitur dari citra input. Konvolusi akan menghasilkan *transformasi linear* dari data *input* sesuai informasi spasial pada data. Bobot pada layer tersebut menspesifikasikan kernel konvolusi yang digunakan, sehingga kernel konvolusi dapat dilatih berdasarkan input pada CNN(Eka Putra, 2016).

2.4.2 *Max pooling*

Pooling layer merupakan cara untuk melakukan pengurangan ukuran *matrix*. *Pooling layer* terdiri dari sebuah filter yang memiliki ukuran dan nilai(Hakim & Rainarli, 2019). Filter ini akan bergeser pada seluruh area *feature map*. Ukuran dan jumlah parameter akan berkurang setelah menggunakan *layer* ini, sehingga akan mempercapat komputasi. *Pooling layer* memiliki beberapa macam tipe antara lain *average pooling*, *max pooling*, dan *Lp Pooling*(Zufar & Setiyono, 2016).



Gambar 2.4 Operasi max pooling

Sumber: (Walters, 2019)

2.4.3 *ReLu*

Fungsi aktivasi untuk menormalisasikan nilai yang dihasilkan dari layer konvolusi yaitu *ReLu*. Fungsi *ReLu* bekerja dengan menormalisasikan nilai sehingga tidak ada nilai dibawah 0 dengan menggunakan fungsi *max*(Achmad et al., 2019).

2.4.4 *Batch normalization*

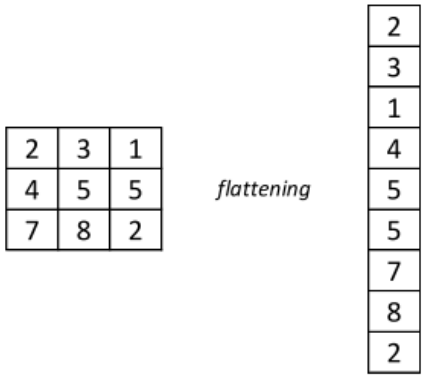
Batch normalization merupakan metode yang dapat diterapkan untuk menormalkan input dari setiap lapisan. Tujuan menggunakan batch normalization untuk mengatasi masalah pergeseran *kovariat internal (internal covariate shift*). Manfaat dari *batch normalization* yaitu membantu menstabilkan pelatihan dan sangat efektif mengurangi *epoch* yang dibutuhkan untuk melatih jaringan saraf(Astuti, 2019).

2.4.5 *Dropout*

Tujuan dari dropout yaitu mencegah terjadi *overfitting* dan mempercepat proses learning. *Overfitting* merupakan kondisi dimana terjadi ketidaksesuaian pada proses prediksi setelah semua data melalui proses training dengan presentase yang baik. Cara kerja *dropout* dengan menghilangkan sementara suatu neuron pada *Hidden Layer* maupun *Visible Layer* yang berada didalam jaringan (Santoso & Ariyanto, 2018).

2.4.6 *Flatten*

Flattening merupakan operasi yang mengubah matriks menjadi vektor satu dimensi. Proses *flattening* mengubah *feature map* yang telah diperoleh dari *layer* sebelumnya menjadi vektor satu dimensi agar *feature map* tersebut dapat diklasifikasikan dengan *fully-connected layer* dan *softmax*. Proses *flatten* diilustrasikan pada Gambar 2.5. Pada tahap *backpropagation* layer ini mengubah vektor satu dimensi kembali menjadi matriks dengan dimensi seperti semula untuk selanjutnya dapat dilakukan proses perubahan bobot filter(Achmad et al., 2019).



Gambar 2.5 Ilustrasi flattern
Sumber: (Achmad et al., 2019)

2.4.7 *Softmax*

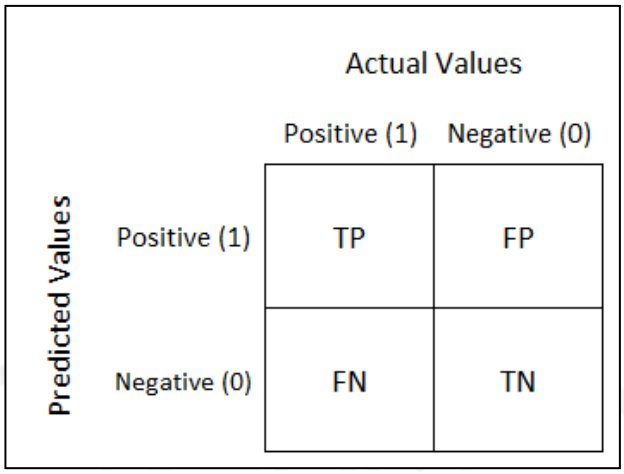
Softmax yaitu fungsi aktivasi yang terdapat pada *output layer*. *Output layer* memiliki perbedaan yang sedikit dengan *fully connected layer*, yang membedakan kedua *layer* ini adalah penggunaan fungsi aktivasi *softmax* pada *layer output* dan fungsi aktivasi *ReLU* pada *fully-connected layer*(Achmad et al., 2019).

2.4.8 *Fully Connected Layer*

Sebelum masuk ke *fully connected layer*, *feature map* akan melalui tahap "*flatten*" atau reshape. Proses *flatten* menghasilkan sebuah vektor yang akan digunakan sebagai *input* dari *fully connected layer*. *Fully connected layer* bertujuan untuk melakukan transformasi pada dimensi data agar data dapat diklasifikasikan secara linear. *Fully connected layer* hanya dapat di implementasikan di akhir jaringan karena dapat menyebabkan data kehilangan informasi spasialnya dan tidak *reversibel*(Eka Putra, 2016).

2.5 Confusion matrix

Confusion matrix merupakan suatu metode yang sering digunakan untuk melakukan perhitungan akurasi. *Confusion matrix* dapat digambarkan menggunakan tabel yang menyatakan jumlah data uji yang benar diklasifikasikan dan jumlah data uji yang salah diklasifikasikan. Tabel pada confussion matrix terdiri dari empat kombinasi berbeda dari nilai prediksi dan nilai aktual(Rahman et al., 2017).



Gambar 2.6 Tabel confussion matrix

Sumber: (Stazio et al., 2019)

Pada Gambar 2.6 dapat dijekasjan dengan True positive (TP) yaitu jumlah data yang kelas aktualnya kelas positif dan kelas prediksinya kelas positif. Dapat di interpretasikan dengan memprediksi positif dan kenyataannya benar. False negative (FN) yaitu jumlah data yang kelas aktualnya kelas positif dan kelas prediksinya kelas negatif. Dapat di interpretasikan dengan memprediksi negatif dan kenyataannya salah. False positive (FP) yaitu jumlah data yang kelas aktualnya kelas negatif dan kelas prediksinya kelas positif. Dapat di interpretasikan dengan memprediksi positif dan kenyataannya salah. True negative (TN) yaitu jumlah data yang kelas aktualnya kelas negatif dan kelas prediksinya kelas negatif. Dapat di interpretasikan dengan memprediksi negatif dan kenyataannya benar(Narkhede, 2018).

A. Akurasi

Akurasi merupakan tingkat kedekatan antara nilai prediksi dengan nilai aktual(Hendriyana & Maulana, 2020). Untuk penghitungan dapat dilihat pada Persamaan (2.1).

$$Akurasi = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.1)$$

B. Presisi

Presisi adalah tingkat ketepatan antara informasi yang diminta oleh pengguna dengan jawaban yang diberikan oleh sistem(Hendriyana & Maulana, 2020). Untuk penghitungan dapat dilihat pada Persamaan (2.2).

$$Presisi = \frac{TP}{TP + FP} \quad (2.2)$$

C. Akurasi

Recall adalah tingkat keberhasilan sistem dalam menemukan kembali sebuah informasi(Hendriyana & Maulana, 2020). Untuk penghitungan dapat dilihat pada Persamaan (2.3).

$$Recall = \frac{TP}{TP + FN} \quad (2.3)$$

2.6 Penelitian sebelumnya

Penelitian sebelumnya yang dilakukan oleh Sang et al(2017) memiliki arsitektur model dengan Menggabungkan BK-Start dan VGG menjadi BKVGG8, BKVGG10, BKVGG12, dan BKVGG14 dalam penelitiannya. Penelitian ini menggunakan dataset FER2013 memiliki akurasi sebesar 71%. Pada penelitian ini emosi dengan akurasi paling tinggi terdapat pada emosi senang.

Penelitian Nugroho et al(2020) memiliki nilai akurasi sebesar 65%. Penelitian ini tidak menuliskan arsitektur modelnya. Emosi dengan akurasi paling tinggi pada penelitian ini terdapat pada emosi senang. Penelitian dari Septian et al, memiliki tingkat akurasi yang hampir sama dengan penelitian Nugroho et al(2020). Penelitian Septian et al memiliki akurasi 64,45% dan emosi dengan akurasi paling tinggi ada di emosi senang.

Penelitian Ramdhani et al(2018) memiliki akurasi sebesar 63,41% tidak jauh berbeda dengan dua penelitian sebelumnya. Dataset yang digunakan dalam penelitian ini yaitu FER2013. Emosi dengan akurasi paling tinggi terdapat pada emosi senang. Arsitektur pada penelitian ini memiliki 9 lapisan. Penelitian Achmad et al(2019) memiliki akurasi 80,75%. Penelitian ini memiliki akurasi paling tinggi dari penelitian lainnya. Dataset yang digunakan yaitu CK+. Untuk ringkasan dari penelitian sebelumnya dapat dilihat pada Tabel 2.1.

Tabel 2.1 Penelitian yang menggunakan CNN

No	Judul	Penulis	Dataset	Akurasi
1.	Facial Expression Recognition Using Deep Convolutional Neural Network(Sang et al., 2017)	Dinh Viet Sang, Nguyen Van Dat, & Do Phan Thuan	FER2013	71
2.	Implementasi Deep Learning Menggunakan Convolutional Neural	Pulung Adi Nugroho, Indah	FER2013	65

	Network (CNN) pada Ekspresi Manusia(Nugroho et al., 2020)	Fenriana, & Rudy Arijanto, M.Kom		
3.	Klasifikasi Emosi Menggunakan Convolutional Neural Networks (Septian et al., n.d.)	Ripan Septian, Dede Irawan Saputra, & Susanto Sambari	FER2013	64,54
4.	Convolutional Neural Networks Models for Facial Expression Recognition(Ramdhani et al., 2018)	Burhanudin Ramdhani, Esmeralda C. Djamal, & Ridwan Ilyas	FER2013	63,41
5.	Klasifikasi Emosi Berdasarkan Ciri Wajah Menggunakan Convolutional Neural Network(Achmad et al., 2019)	Acmad Yusuf, Randy Cahya Wihandika, & Chandra Dewi	CK+	80,75

BAB III
METODOLOGI

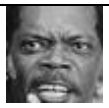
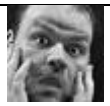
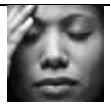
3.1 Data

Penelitian ini menggunakan data citra emosi yang telah dilabeli sebelumnya. Data tersebut berlabel emosi-emosi yang akan digunakan untuk membangun model pada penelitian ini. Emosi yang digunakan dalam penelitian ini antara lain, marah, sedih, senang, natural, jijik, takut, dan terkejut. Data pelatihan akan digunakan dalam pelatihan hingga terbentuknya model sedangkan untuk data uji akan digunakan untuk mengetahui keakuratan dari model yang telah dibuat. Penelitian ini menggunakan tiga dataset citra emosi yaitu, Facial Expression Recognition 2013(FER2013), Cohn-Kanade Dataset (CK+), dan Karolinska Directed Emotional Faces (KDEF).

Data FER2013 merupakan dataset yang digunakan dalam kompetisi yang pernah diadakan oleh salah satu website yang bernama Kaggel. Kompetisi tersebut bernama "Challenges in Representation Learning: Facial Expression Recognition Challenge"(Carrier & Courville, 2013). Data yang digunakan dari kompetisi tersebut dipersiapkan oleh Pierre-Luc Carrier dan Aaron Courville. Data FER2013 sudah terbagi antara data pelatihan dan data uji dengan perbandingan 80% dan 20%. Data pelatihan berisikan 28.709 citra dan untuk data uji berisikan 7178 citra. Pada FER2013, jumlah citra pada masing-masing emosi berbeda-beda. Citra terbanyak terdapat pada emosi senang dengan jumlah 8989 citra sedangkan citra paling sedikit terdapat pada emosi jijik dengan jumlah 547 citra. Data tersebut berukuran 48 x 48 pixels. Citra emosi tersebut berwarna greyscale atau hanya berwarna abu-abu dan putih. Data FER2013 berisikan jenis kelamin laki-laki dan perempuan tetapi pada data ini tidak diketahui jumlah antara masing-masing jenis kelamin. Ras pada data ini juga tidak disebutkan oleh pembuat. Untuk umur orang pada data ini tidak diketahui tetapi kita dapat melihat jika ada dari umur bayi hingga lanjut usia. Tabel 3.1 menunjukkan contoh citra dari dataset FER2013.

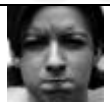
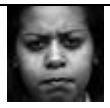
Tabel 3.1 Contoh dataset FER 2013

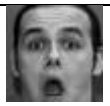
Emosi	**Contoh citra**				
Neutral					

Angry					
Disgust					
Fear					
Happy					
Sad					
Surprise					

Data CK+ berisikan citra emosi yang diambil dari 210 objek dengan rentang usia dari 18 hingga 50 tahun. Data tersebut memiliki perbandingan laki-laki 31% dan perempuan 69%. Ras yang terdapat pada CK+ yaitu, 81% *Euro-American*, 13% *Afro-American*, dan 6% golongan lain. Data CK+ berjumlah 981 citra dengan jumlah masing-masing kelas yang berbeda-beda. Data dengan jumlah paling sedikit terdapat pada kelas netral dengan jumlah 54 citra dan paling banyak terdapat pada kelas *surprise* dengan jumlah 249 citra. Data CK+ berwarna *greyscale* dengan ukuran 48x48 *pixels*. Tabel 3.2 menunjukkan contoh citra dari dataset CK+.

Tabel 3.2 Contoh dataset CK+

Emosi	**Contoh citra**				
Neutral					
Angry					
Disgust					
Fear					

Happy					
Sad					
Surprise					

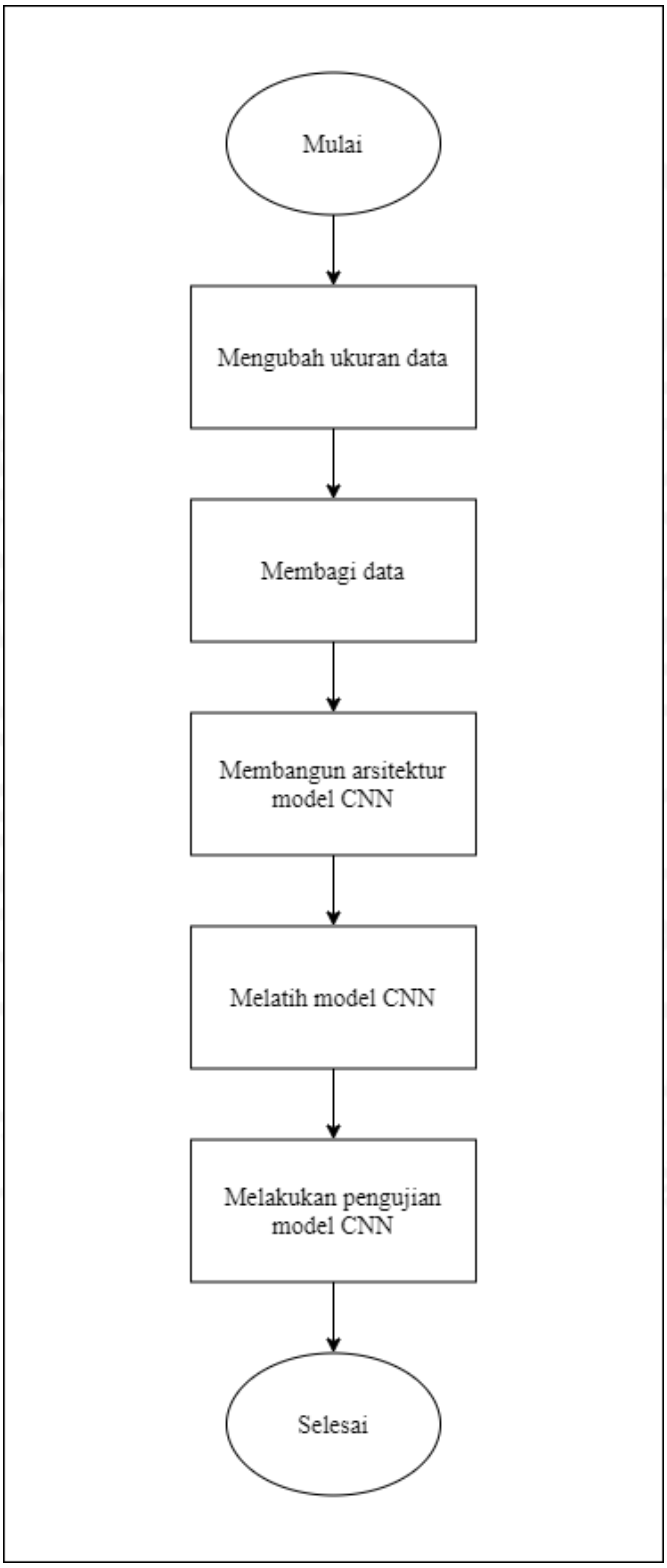
Dataset KDEF berisikan 490 citra emosi dengan jumlah masing-masing kelas rata dengan jumlah 70 citra. Dataset ini diambil dari 35 orang laki-laki dan 35 orang perempuan dengan rentang usia 20 hingga 30 tahun. Dataset KDEF berwarna *greyscale* dan memiliki ukuran 256x256 *pixels*. Tabel 3.3 menunjukkan contoh citra dari dataset KDEF,

Tabel 3.3 Contoh dataset KDEF

Emosi	**Contoh citra**				
Neutral					
Angry					
Disgust					
Fear					
Happy					
Sad					
Surprise					

3.2 Perancangan penelitian

Perancangan penelitian menjelaskan alur program pada penelitian ini. Alur tersebut akan digambarkan dengan alur *flowchart*. Gambar 3.1 merupakan gambar *flowchart* untuk perancangan penelitian.



Gambar 3.1 *Flowchart* perancangan penelitian

Dari Gambar 3.1 Proses mengubah ukuran data dan membagi data termasuk ke dalam proses pre-processing dan untuk membangun arsitektur model hingga melakukan pengujian model termasuk kedalam proses pada metode CNN.

3.2.1 Mengubah Ukuran Data

Ukuran data pada dataset KDEF adalah 256x256 *pixels*. Penelitian ini menggunakan dataset dengan ukuran 48x48 *pixels* sehingga diperlukan pengubahan ukuran agar *dataset* tersebut dapat digunakan. Untuk dataset CK+ dan FER memilki ukuran 48x48 *pixel* sehingga tidak diperlukan adanya pengubahan ukuran. Pengubahan ukuran dilakukan hanya dengan mengurangi *pixel* tanpa memotong citra.

3.2.2 Membagi Data

Pembagian data yang pertama dilakukan untuk menyamakan jumlah dataset dari masing-masing kelas emosi yang berasal dari gabungan dari tiga *dataset* yang digunakan. Setelah menyamakan jumlah masing-masing kelas akan dilakukan lagi pembagian dataset dengan rasio 80% sebagai data latih, 10% data uji, dan 10% data validasi. Data latih akan digunakan untuk melakukan pelatihan model, data uji digunakan untuk melakukan pengujian dari model yang telah terlatih, dan untuk data validasi akan digunakan pada saat pelatihan. Tabel 3.4 menunjukkan jumlah *dataset* awal,

Tabel 3.4 Dataset awal

Dataset	**Emosi**							**Total**
	Netral	***Angry***	***Disgust***	***Fear***	***Happy***	***Sad***	***Surprise***	
KDEF	70	70	70	70	70	70	70	490
FER2013	6193	4953	547	5121	8989	6077	4002	35882
CK+	54	135	177	75	207	84	249	981
Total	6317	5158	794	5266	9266	6231	4321	37353

Dan tabel 3.5 menunjukkan jumlah data setelah dibagi dan disamakan jumlah nya,

Tabel 3.5 Dataset yang akan digunakan

Dataset	**Emosi**							**Total**
	Netral	**Angry**	**Disgust**	**Fear**	**Happy**	**Sad**	**Surprise**	
KDEF	70	70	70	70	70	70	70	490
FER2013	581	561	547	581	537	578	523	3908
CK+	18	45	59	25	69	28	83	327

TOTAL	676	676	676	676	676	676	676	4732

Berikut alur untuk menyamakan =

i. Mengeliminasi data CK+ yang terdapat duplikasi pada masing-masing kelas dari satu objek yang sama.
ii. Mengambil seluruh kelas paling kecil dari dataset FER yaitu dari kelas *fear*, karena pada dataset ini dataset jumlah masing-masing kelas tidak seimbang dan perbedaan nya tinggi.
iii. Mengambil seluruh dataset pada KDEF karena jumlah dataset masing-masing kelas sudah sama dan tidak ada duplikasi.
iv. Menjumlahkan ketiga dataset yang ada di kelas *fear* dan mendapatkan jumlah akhir 676 data untuk satu kelas.
v. Menyamakan seluruh kelas dataset dengan jumlah 676 citra kemudian mengurangi total *dataset* gabungan dari CK+ dan KDEF untuk mengambil jumlah pada *dataset* FER.

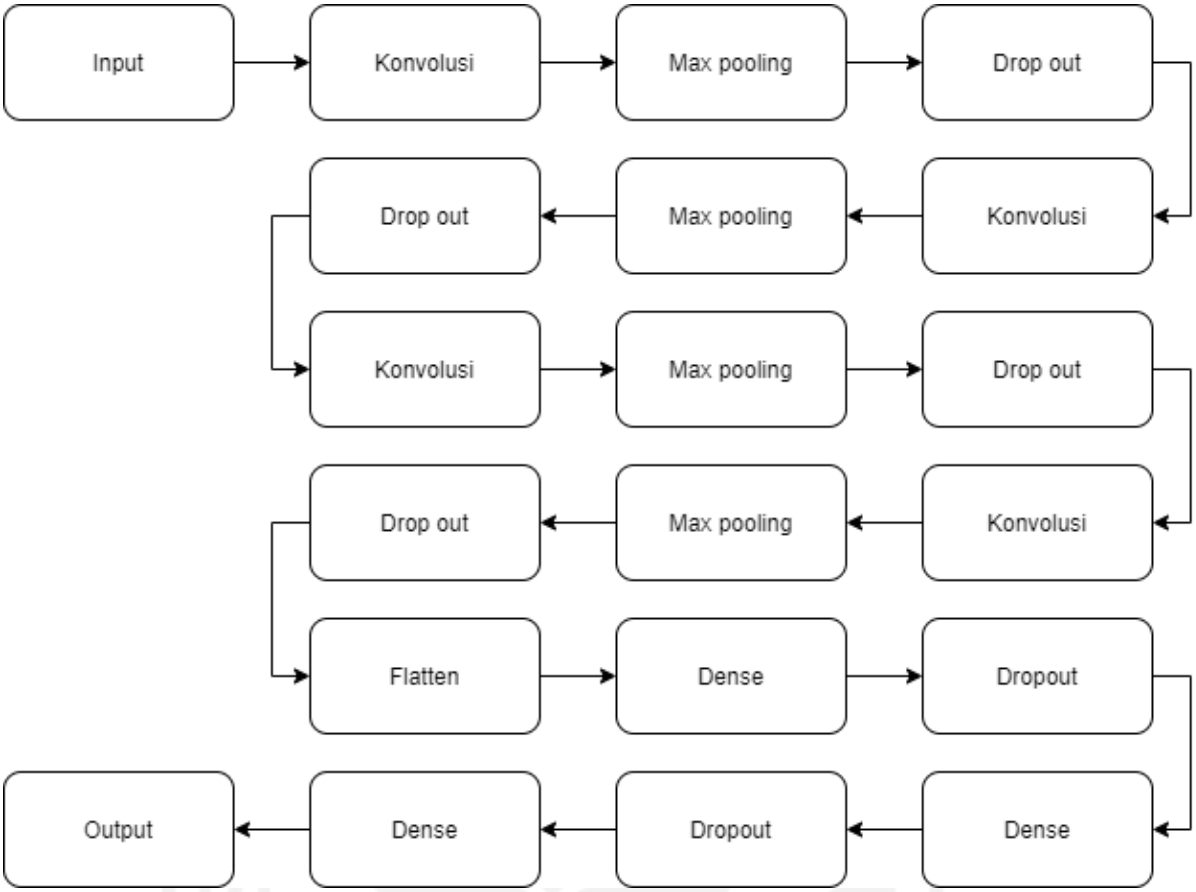
Setelah dataset memiliki jumlah yang sama kemudian dibagi lagi menjadi data latih, data uji, dan data validasi. Tabel 3.6 menunjukkan jumlah hasil pembagiannya,

Tabel 3.6 Pembagian *dataset*

Kategori	**Dataset**			**Jumlah data**
	FER2013	**KDEF**	**CK+**	
Data latih	3132	392	263	3787
Data validasi	391	49	33	473
Data uji	392	49	33	474
Total	3915	490	329	4734

3.2.3 Membangun arsitektur model

Arsitektur yang akan dibangun terdiri dari beberapa lapisan yang dapat dilihat pada Gambar 3.2. Terdapat lapisan konvolusi dan lapisan *fully connected* pada arsitektur ini. Pada lapisan konvolusi berisi fungsi aktivasi *ReLu, Max pooling, Batch normalization,* dan *Dropout*. Lapisan *fully connected* berisikan *Flatten, Dense, Batch normalization,* dan *Softmax*.



Gambar 3.2 Arsitektur CNN

Lapisan pertama pada arsitektur ini yaitu layer konvolusi dengan filter sebanyak 64 dan memiliki ukuran *kernel* 3x3. Pada lapisan ini terdapat aktivasi *ReLu, Max pooling, Batch normalization,* dan terdapat *dropout*. Lapisan kedua hingga lapisan keempat hampir sama dengan lapisan pertama tetapi memiliki sedikit perbedaan. Perbedaan itu sebagai berikut pada lapisan kedua filter yang digunakan sebanyak 128 dengan ukuran *kernel* 5x5 dan pada lapisan ketiga dan keeempat *filter* yang digunakan 512 dengan ukuran *kernel* 3x3. Setelah lapisan keempat ditutup dengan *Flatten.*

Lapisan berikutnya terdapat lapisan *fully connected* sebanyak 2 lapisan. Lapisan *fully connected* terdapat *Dense*. Kemudian *Batch normalization*, aktivasi *Relu*, dan juga *Dropout*. Terakhir menggunakan fungsi aktivasi *softmax* untuk menentukan hasil dari deteksi emosi.

3.2.4 Melatih model

Setelah arsitektur terbentuk kemudian dilakukan pelatihan model. Pelatihan model dijalankan dengan beberapa scenario untuk mendapatkan hasil yang terbaik. Pelatihan dengan optimasi yang berbeda-beda yaitu, Adam, RMS, dan SGD dengan epoch 200. Setelah

mendapatkan hasil dengan optimasi terbaik kemudian dilakukan dengan perbedaan epoch dalam melakukan pelatihan.

3.2.5 Melakukan pengujian model

Pengujian dilakukan pada model yang telah dilakukan pelatihan. Pengujian dilakukan dengan *dataset* FER, CK+, KDEF, dan gabungan dari ketiga *dataset*. Hasil dari pengujian adalah akurasi, presisi, dan *recall*.

BAB IV
HASIL DAN PEMBAHASAN

4.1 Implementasi

Semua tahapan perancangan menggunakan *flowchart* seperti pada Gambar 3. 1. diimplementasikan dalam kode program dengan Bahasa pemrograman *Python versi 3.7* dijalankan dengan bantuan *software jupyter notebook*. langkah-langkah dan kode program untuk setiap proses yang ada pada sistem akan dijelaskan pada Hasil dan Pembahasan ini. Pada awal dilakukan *import package-package* yang akan digunakan. Berikut kode untuk *import package*,

```
    import numpy as np
    import pandas as pd
    import cv2
    import seaborn as sns
    import matplotlib.pyplot as plt
    from sklearn import metrics
    from sklearn.metrics import confusion_matrix
    from keras.utils import np_utils
    from keras.layers import
Dense,Input,Dropout,GlobalAveragePooling2D,Flatten,Conv2D,BatchNormalizatio
n,Activation,MaxPooling2D
    from keras.models import Model,Sequential
    from keras.optimizers import Adam,SGD,RMSprop
```

Gambar 4.1 Kode *import package*

Packages numpy digunakan dalam operasi komputasi tipe data numerik dan dapat diterapkan pada vector dan matriks. Kemudian packages *pandas* menyediakan struktur data dan analisis data yang mudah digunakan. Struktur data dasar pada *Pandas* dinamakan *DataFrame* yang memudahkan dalam pembabacaan sebuah *file* dengan banyak jenis format. *Pandas* juga digunakan untuk membuat tabel, mengubah dimensi data, dan mengecek data.

Packages *cv2* digunakan untuk mengolah gambar. Dalam packages ini terdiri untuk membaca gambar, menampilkan gambar, dan menyimpan gambar. Packages *matplotlib* merupakan pustaka visualisasi data *multiplatform* yang dibangun di atas *array NumPy* dan dapat digunakan untuk memvisualisasikan data secara 2D maupun 3D. Packages *sklearn* untuk menerapkan pembelajaran mesin standar dan tugas-tugas penambangan data seperti

mengurangi dimensi, klasifikasi, regresi, pengelompokan data, dan pemilihan model. Package *keras* digunakan untuk mengembangkan model *deep learning* karena metode yang dalam penelitian ini adalah CNN. *Keras* dapat dijalankan di atas *framework* (kerangka kerja) kecerdasan buatan seperti *TensorFlow, Microsoft Cognitive Toolkit*, dan *Theano*. Tabel 1111 Berisi nama packages dan kegunaan pada penelitian ini.

Tabel 4.1 Packages dan kegunaannya.

No	Packages	Kegunaan
1.	Numpy	Pemrosesan matriks karena data citra akan menjadi sebuah matriks
2.	Pandas	Membuat data frame yang terdiri dari data informasi citra dan data label citra
3.	OpenCV	Pembacaan data citra
4.	Matplotlib	Membuat plot dari hasil pelatihan
5.	Keras	Untuk menggunakan CNN

4.1.1 Mengubah ukuran data

Dataset KDEF berukuran 256 x 256 *pixels* sehingga tidak dapat diproses karena masukan untuk model penelitian ini berukuran 48 x 48 *pixels*. Sehingga diperlukan pengubahan ukuran pada *dataset* KDEF. Gambar 4.2 digunakan untuk mengubah ukuran dari 256x256 *pixels* menjadi 48x48 *pixels* tanpa memotong gambar asli data tersebut.

```
   path0 = 'E:/TA KU/Dataset/kdef/0'
   path1 = 'E:/TA KU/Dataset/kdef/1'
   path2 = 'E:/TA KU/Dataset/kdef/2'
   path3 = 'E:/TA KU/Dataset/kdef/3'
   path4 = 'E:/TA KU/Dataset/kdef/4'
   path5 = 'E:/TA KU/Dataset/kdef/5'
   path6 = 'E:/TA KU/Dataset/kdef/6'

   images0 = [os.path.join(path0,f) for f in os.listdir(path0) if
os.path.isfile(os.path.join(path0,f))]
   images1 = [os.path.join(path1,f) for f in os.listdir(path1) if
os.path.isfile(os.path.join(path1,f))]
```

```
    images2 = [os.path.join(path2,f) for f in os.listdir(path2) if
os.path.isfile(os.path.join(path2,f))]
    images3 = [os.path.join(path3,f) for f in os.listdir(path3) if
os.path.isfile(os.path.join(path3,f))]
    images4 = [os.path.join(path4,f) for f in os.listdir(path4) if
os.path.isfile(os.path.join(path4,f))]
    images5 = [os.path.join(path5,f) for f in os.listdir(path5) if
os.path.isfile(os.path.join(path5,f))]
    images6 = [os.path.join(path6,f) for f in os.listdir(path6) if
os.path.isfile(os.path.join(path6,f))]

    width,height = 48,48

    i = 0
    while i < 70:
        pth = images0[i]
        img = cv2.imread(pth)
        imgresize = cv2.resize(img, (width, height))
        new_path = '%s/kdef00%s.png' % (path0, i)
        io.imsave(new_path, imgresize)
        i += 1
```

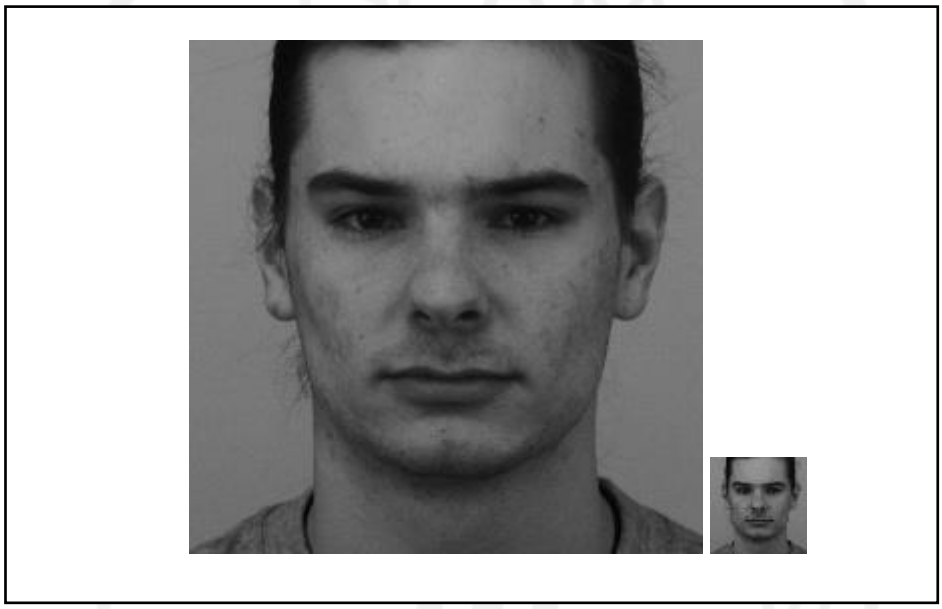
Gambar 4.2 Kode untuk mengubah ukuran data KDEF

Kode pada Gambar 4.2 dilakukan pertama-tama dengan menyimpan lokasi *file* yang akan digunakan berada, lokasi tersebut disimpan dengan variabel *path*. Untuk penjelasan emosi pada folder dapat dilihat pada Tabel 4.2.

Tabel 4.2 Path folder emosi

Path folder	Emosi
0	Netral
1	Marah
2	Jijik
3	Takut
4	Senang
5	Sedih
6	Terkejut

Variabel *width* dan *height* menyimpan angka 48 dengan tujuan untuk menjadi tujuan ukuran setelah diubah. Menggunakan perulangan sebanyak enam kali dikarenakan terdapat 7 *folder* emosi. Pada Gambar 4.2 perulangan hanya ditampilkan pada *folder* 0. Perbedaan perulangan dari perulangan pada gambar 4.2 dengan perulangan yang tidak dituliskan hanya pada *path* nya saja. Pengubahan ukuran dengan menggunakan fungsi *resize()*. Setelah ukuran berubah kemudian disimpan dengan menggunakan fungsi *imsave()*. Contoh hasil dari perubahan ukuran dapat dilihat pada gambar 4.3 dengan citra kiri sebelum dilakukan perubahan ukuran dan citra yang kanan setelah mendapatkan perubahan ukuran.



Gambar 4.3 Perbedaan ukuran citra KDEF

4.1.2 Membagi data

Kode program pada Gambar 4.4 hingga Gambar 4.10 bertujuan untuk mengakses data set yang berupa gambar dengan format PNG. Pengaksesan data mengambil data dari folder-folder yang telah terpisah antara folder jenis dataset (data latih, data uji, dan data validasi) dan *folder* emosi. Penjelasan *folder* dapat dilihat pada Tabel 4.1.

```
loc1 = 'E:/TA KU/Dataset/Gabungan/TRAIN/0'
loc2 = 'E:/TA KU/Dataset/Gabungan/TRAIN/1'
loc3 = 'E:/TA KU/Dataset/Gabungan/TRAIN/2'
loc4 = 'E:/TA KU/Dataset/Gabungan/TRAIN/3'
loc5 = 'E:/TA KU/Dataset/Gabungan/TRAIN/4'
loc6 = 'E:/TA KU/Dataset/Gabungan/TRAIN/5'
loc7 = 'E:/TA KU/Dataset/Gabungan/TRAIN/6'
```

Gambar 4. 4 Kode untuk menyimpan *path* dari lokasi data latih

Kode pada Gambar 4.4 mengambil data latih pada folder gabungan. Untuk penjelasan angka dari masing-masing folder dapat dillihat ada Tabel 4.1. Setelah menyimpan *path* dari masing-masing folder emosi kemudian dilakukan pembacaan dan membuat label.

```
labels = []
for i in os.listdir(loc1):
    labels.append(0)
for i in os.listdir(loc2):
    labels.append(1)
for i in os.listdir(loc2):
    labels.append(2)
for i in os.listdir(loc2):
    labels.append(3)
for i in os.listdir(loc2):
    labels.append(4)
for i in os.listdir(loc2):
    labels.append(5)
for i in os.listdir(loc2):
    labels.append(6)

features = []
from tqdm import tqdm
for i in tqdm(os.listdir(loc1)):
    features.append(cv2.imread(os.path.join(loc1,i),0))
for i in tqdm(os.listdir(loc2)):
    features.append(cv2.imread(os.path.join(loc2,i),0))
for i in tqdm(os.listdir(loc3)):
    features.append(cv2.imread(os.path.join(loc3,i),0))
for i in tqdm(os.listdir(loc4)):
    features.append(cv2.imread(os.path.join(loc4,i),0))
for i in tqdm(os.listdir(loc5)):
    features.append(cv2.imread(os.path.join(loc5,i),0))
for i in tqdm(os.listdir(loc6)):
    features.append(cv2.imread(os.path.join(loc6,i),0))
for i in tqdm(os.listdir(loc7)):
    features.append(cv2.imread(os.path.join(loc7,i),0))
```

Gambar 4.5 Kode untuk membuat label dan membaca data latih.

Pada Gambar 4.5 variabel labels digunakan untuk membuat label sesuai jumlahnya dengan isi dari folder-folder emosi yang ada di *training*.Untuk menambahkan label sesuai dengan nama folder nya digunakan fungsi append() yang berfungsi untuk menambahkan item baru di belakang list. *Features* membaca data dari citra yang berformat jpg dengan menggunakan fungsi *imread()* yang terdapat pada *OpenCv*. Pembacaan citra akan menjadi matriks ketika sudah masuk ke dalam variabale features.

```
loc1 = 'E:/TA KU/Dataset/Gabungan/TEST/GABUNGAN/0'
loc2 = 'E:/TA KU/Dataset/Gabungan/TEST/GABUNGAN/1'
loc3 = 'E:/TA KU/Dataset/Gabungan/TEST/GABUNGAN/2'
loc4 = 'E:/TA KU/Dataset/Gabungan/TEST/GABUNGAN/3'
loc5 = 'E:/TA KU/Dataset/Gabungan/TEST/GABUNGAN/4'
loc6 = 'E:/TA KU/Dataset/Gabungan/TEST/GABUNGAN/5'
loc7 = 'E:/TA KU/Dataset/Gabungan/TEST/GABUNGAN/6'
```

Gambar 4. 6 Kode untuk menyimpan *path* dari lokasi data uji gabungan.

Kode pada Gambar 4.6 menyimpan *path* data uji pada folder gabungan. Untuk penjelasan angka pada *folde*r dapat dilihat pada Tabel 4.1. Setelah melakukan penyimpanan dilakukan hal yang sama dengan Gambar 4.5 dengan memebuat label dan membaca data citra pada *folder* data uji.

```
loc1 = 'E:/TA KU/Dataset/Gabungan/VALIDATION/GABUNGAN/0'
loc2 = 'E:/TA KU/Dataset/Gabungan/VALIDATION/GABUNGAN/1'
loc3 = 'E:/TA KU/Dataset/Gabungan/VALIDATION/GABUNGAN/2'
loc4 = 'E:/TA KU/Dataset/Gabungan/VALIDATION/GABUNGAN/3'
loc5 = 'E:/TA KU/Dataset/Gabungan/VALIDATION/GABUNGAN/4'
loc6 = 'E:/TA KU/Dataset/Gabungan/VALIDATION/GABUNGAN/5'
loc7 = 'E:/TA KU/Dataset/Gabungan/VALIDATION/GABUNGAN/6'
```

Gambar 4.7 Kode untuk menyimpan *path* dari lokasi data validasi.

Kode pada Gambar 4.7 menyimpan *path* data validasi pada folder gabungan. Untuk penjelasan angka pada *folde*r dapat dilihat pada Tabel 4.1. Setelah melakukan penyimpanan dilakukan hal yang sama dengan Gambar 4.5 dengan memebuat label dan membaca data citra pada *folder* data uji.

```
loc1 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/CK+/0'
```

```
loc2 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/CK+/1'
loc3 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/CK+/2'
loc4 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/CK+/3'
loc5 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/CK+/4'
loc6 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/CK+/5'
loc7 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/CK+/6'
```

Gambar 4. 8 Kode untuk menyimpan *path* dari lokasi data uji CK+.

Kode pada Gambar 4.8 menyimpan *path* data uji CK+ pada folder gabungan. Untuk penjelasan angka pada *folde*r dapat dilihat pada Tabel 4.1. Setelah melakukan penyimpanan dilakukan hal yang sama dengan Gambar 4.5 dengan memebuat label dan membaca data citra pada *folder* data uji.

```
loc1 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/KDEF/0'
loc2 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/KDEF/1'
loc3 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/KDEF/2'
loc4 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/KDEF/3'
loc5 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/KDEF/4'
loc6 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/KDEF/5'
loc7 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/KDEF/6'
```

Gambar 4. 9 Kode untuk menyimpan *path* dari lokasi data uji KDEF.

Kode pada Gambar 4.9 menyimpan *path* data uji KDEF pada folder gabungan. Untuk penjelasan angka pada *folde*r dapat dilihat pada Tabel 4.1. Setelah melakukan penyimpanan dilakukan hal yang sama dengan Gambar 4.5 dengan memebuat label dan membaca data citra pada *folder* data uji.

```
loc1 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/FER/0'
loc2 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/FER/1'
loc3 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/FER/2'
loc4 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/FER/3'
loc5 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/FER/4'
loc6 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/FER/5'
loc7 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/FER/6'
```

Gambar 4.10 Kode untuk menyimpan *path* dari lokasi data uji FER2013.

Kode pada Gambar 4.10 menyimpan *path* data uji FER2013 pada folder gabungan. Untuk penjelasan angka pada *folde*r dapat dilihat pada Tabel 4.1. Setelah melakukan

penyimpanan dilakukan hal yang sama dengan Gambar 4.5 dengan memebuat label dan membaca data citra pada *folder* data uji.

```
train_data = pd.DataFrame()
train_data['emotion'] = labels
train_data['pixel_values'] = features

test_data = pd.DataFrame()
test_data['emotion'] = labels_test
test_data['pixel_values'] = test_features

val_data = pd.DataFrame()
val_data['emotion'] = labels_val
val_data['pixel_values'] = val_features

testck_data = pd.DataFrame()
testck_data['emotion'] = labels_testck
testck_data['pixel_values'] = testck_features

testkdef_data = pd.DataFrame()
testkdef_data['emotion'] = labels_testkdef
testkdef_data['pixel_values'] = testkdef_features

testfer_data = pd.DataFrame()
testfer_data['emotion'] = labels_testfer
testfer_data['pixel_values'] = testfer_features
```

Gambar 4. 11 Kode untuk membuat data frame

```
train_data.head()
```

	emotion	pixel_values
0	0	[[10, 2, 1, 2, 4, 3, 2, 7, 10, 12, 16, 14, 12,...
1	0	[[120, 100, 89, 100, 108, 100, 114, 144, 161, ...
2	0	[[83, 63, 55, 57, 66, 67, 69, 72, 80, 89, 91, ...
3	0	[[82, 72, 69, 65, 55, 60, 69, 92, 84, 75, 64, ...
4	0	[[220, 219, 220, 219, 202, 153, 64, 49, 42, 38...

Gambar 4. 12 Hasil proses kode pada gambar 4.11

Kode pada Gambar 4.11 membuat 6 data frame baru dengan fungsi *DataFrame()* yang terdapat pada *pandas*. Kemudian masing-masing *dataframe* dibuat variabel baru dengan isi dari variabel label yang berisi label emosi dan *feature* yang yang berisikan matriks dari citra yang telah disimpan. Pada Gambar 4.12 dapat dilihat angka pada coloum yang berisi 0 merupakan label emosi 0 yaitu netral dan pada kolom *pixel_values* merupakan matriks dari data informasi citra.

```
#Bagoian 1
features = np.array(features).reshape(-1,48,48,1)
test_features = np.array(test_features).reshape(-1,48,48,1)
val_features = np.array(val_features).reshape(-1,48,48,1)
testck_features = np.array(testck_features).reshape(-1,48,48,1)
testkdef_features = np.array(testkdef_features).reshape(-1,48,48,1)
testfer_features = np.array(testfer_features).reshape(-1,48,48,1)

#Bagian 2
features = features/255
test_features = test_features/255
val_features = val_features/255
testck_features = testck_features/255
testkdef_features = testkdef_features/255
testfer_features = testfer_features/255

#Bagian 3
labels = np_utils.to_categorical(labels)
labels_test =np_utils.to_categorical(labels_test)
labels_val =np_utils.to_categorical(labels_val)
labels_testck =np_utils.to_categorical(labels_testck)
labels_testkdef =np_utils.to_categorical(labels_testkdef)
labels_testfer =np_utils.to_categorical(labels_testfer)
```

Gambar 4. 13 Kode untuk mengubah array data

```
features
```

```
[array([[10,  2,  1, ..., 14, 12, 46],
        [ 4,  1,  1, ..., 14, 11, 30],
        [ 1,  0,  0, ...,  5, 10, 24],
        ...,
        [65, 67, 69, ..., 77, 76, 75],
        [66, 66, 68, ..., 76, 76, 75],
        [68, 68, 68, ..., 76, 76, 76]], dtype=uint8),
```

Gambar 4.14 Isi variabel features sebelum

```
features
```

```
array([[[[0.03921569],
         [0.00784314],
         [0.00392157],
         ...,
         [0.05490196],
         [0.04705882],
         [0.18039216]],

        [[0.01568627],
         [0.00392157],
         [0.00392157],
         ...,
         [0.05490196],
         [0.04313725],
         [0.11764706]],
```

Gambar 4.15 Isi variabel features sesudah

```
print(labels)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

Gambar 4.16 Isi variabel labels sebelum

```
labels
```

```
array([[1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.]], dtype=float32)
```

Gambar 4.17 Isi variabel label sesudah

Kode pada Gambar 4.12 bagian 1 mengubah isi *array* yang ada dengan ukuran 48x48 dan 1 karena gambar yang digunakan *greyscale*. Bagian 2 membagi isi yang terdapat pada *features* dengan 256. Bagian 3 mengubah isi variabel label dari tipe data *integer* menjadi matriks *integer* yang memiliki nilai biner karena akan digunakan dalam pendeteksian emosi citra. Isi variabel sebelum digunakan kode pada Gambar 4.12 dapat dilihat pada Gambar 4.13 dan Gambar 4.15. Untuk sesudahnya dapat dilihat pada Gambar 4.14 dan Gambar 4.16.

4.1.3 Membangun arsitektur model

```
model = Sequential()
#Layer 1
model.add(Conv2D(64,(3,3),padding = 'same',input_shape = (48,48,1)))
```

```
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.25))

#Layer 2
model.add(Conv2D(128,(5,5),padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout (0.25))

#Layer 3
model.add(Conv2D(512,(3,3),padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout (0.25))

#Layer 4
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(7, activation='softmax'))
```

```
    opt = Adam(lr = 0.0001)
    model.compile(optimizer=opt,loss='categorical_crossentropy',
metrics=['accuracy'])
    model.summary()
```

Gambar 4. 18 Kode untuk membangun arsitektur CNN

Kode pada Gambar 4.17 membuat arsitektur model CNN. Model dibuat dengan fungsi *sequential()* sehingga menghasilkan model yang berlapis-lapis. Untuk menambahkan layer menggunakan fungsi *add()*. Arsitektur terdiri dari layer konvolusi, *flatten,* dan *fully connected*.

Layer 1 menggunakan konvolusi dengan filter sebanyak 64. Ukuran *kernel* yang digunakan pada layer 1 yaitu 3x3. *Padding* yang digunakan yaitu *same*. *Layer* 1 terdapat juga *batch* normalisasi, *dropout* sebesar 0.25, dan *maxpooling* dengan dengan ukuran 2x2. Aktivasi yang digunakan pada *layer* 1 yaitu menggunakan *ReLu*.

L*ayer* 2 sampai dengan 4 tidak berbeda jauh dengan *layer* 1. Pada *layer* 2 konvolusi menggunakan filter sebanyak 128 dan menggunakan *kernel* dengan ukuran 5x5. *Layer* 3 dan 4 memiliki kesamaan pada konvolusi dengan filter sebanyak 512 dengan ukuran *kernel* sebesar 3x3. Setelah *layer* konvolusi terdapat *layer fully connected*.

Pada layer fully connected diawali dengan flatten. Kemudian 2 kali dilakukan dengan *dense*, *batch normalization, dropou*t sebesar 0,25 dan aktivasi *ReLu*. Terakhir *fully connected* dengan aktivasi *softmax*.

Optimasi yang digunakan pada arsitektur yaitu Adam dengan *learning rate* sebesar 0,0001 dan dibuat 2 arsitektur lain dengan beda optimasi yaitu menggunakan RMS dan SGD dengan *learning rate* yang sama. Model di susun dengan fungsi *compile()*. *Loss* fungsi yang digunakan yaitu *categorical cross entropy*. Untuk melihat hasil dari model yang dibangun menggunakan fungsi *summary()*. Hasil arsitektur yang dibangun dapat dilihat pada Gambar 4.12,

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 48, 48, 64)        640
_________________________________________________________________
batch_normalization_1 (Batch (None, 48, 48, 64)        256
_________________________________________________________________
activation_1 (Activation)    (None, 48, 48, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 64)        0
_________________________________________________________________
```

dropout_1 (Dropout)	(None, 24, 24, 64)	0
conv2d_2 (Conv2D)	(None, 24, 24, 128)	204928
batch_normalization_2 (Batch	(None, 24, 24, 128)	512
activation_2 (Activation)	(None, 24, 24, 128)	0
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 128)	0
dropout_2 (Dropout)	(None, 12, 12, 128)	0
conv2d_3 (Conv2D)	(None, 12, 12, 512)	590336
batch_normalization_3 (Batch	(None, 12, 12, 512)	2048
activation_3 (Activation)	(None, 12, 12, 512)	0
max_pooling2d_3 (MaxPooling2	(None, 6, 6, 512)	0
dropout_3 (Dropout)	(None, 6, 6, 512)	0
conv2d_4 (Conv2D)	(None, 6, 6, 512)	2359808
batch_normalization_4 (Batch	(None, 6, 6, 512)	2048
activation_4 (Activation)	(None, 6, 6, 512)	0
max_pooling2d_4 (MaxPooling2	(None, 3, 3, 512)	0
dropout_4 (Dropout)	(None, 3, 3, 512)	0
flatten_1 (Flatten)	(None, 4608)	0
dense_1 (Dense)	(None, 256)	1179904
batch_normalization_5 (Batch	(None, 256)	1024
activation_5 (Activation)	(None, 256)	0
dropout_5 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 512)	131584
batch_normalization_6 (Batch	(None, 512)	2048
activation_6 (Activation)	(None, 512)	0

```
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 3591
=================================================================
Total params: 4,478,727
Trainable params: 4,474,759
Non-trainable params: 3,968
_________________________________________________________________
```

Gambar 4.19 Arsitektur model

Dari Gambar 4.18 diketahui parameter yang terdapat pada arsitektur tersebut berjumlah 4.478.727. Untuk parameter yang dapat pelatihan 4.474.759 dan paramter tidak dapat dilatih berjumlah 3.968.

4.1.4 Melatih model

Pelatihan model dilakukan setelah membuat model CNN. Pelatihan ini menggunakan *epoch* sebanyak 200. Berikut baris code dalam melakukan pelatihan,

```
epochs = 200
history = model.fit(x=features,
                    y=labels,
                    batch_size=64,
                    epochs=epochs,
                    validation_data = (val_features,labels_val)
                   )
```

Gambar 4.20 Kode pelatihan model

Baris kode pada Gambar 4.15, untuk menjalankan pelatihan model digunakan fungsi *fit().* perkembangan pelatihan model disimpan dalam *variable history*. Variabel x dan y merujuk pada data yang akan digunakan dalam pelatihan. Variabel x merujuk pada isi matriks data pelatihan dan y merujuk pada label dari data pelatihan. Pelatihan ini menggunakan *batch_size* 64 dan *epoch* sebanyak 200. *Validation_data* merujuk pada data validasi yang digunakan pada pelatihan model ini. Pada pelatihan model ini dapat diketahui perkembangan *val_accuracy, accuracy, val_loss,* dan loss.. Berikut tabel *progres* pelatihan model per 10 *epoch,*

Tabel 4.3 Progres pelatihan dengan optimasi ADAM

Epoch	Loss	Akurasi	Validate loss	validate akurasi
1	2,2896	0,1584	1,9662	0,1429
10	1,8246	0,2973	2,1075	0,1535
20	1,0542	0,4164	1,7153	0,3731
30	1,9293	0,4935	1,5432	0,4563
40	1,1363	0,5749	1,4971	0,4797
50	0,9505	0,6364	1,4613	0,4925
60	0,7659	0,7193	1,4654	0,5032
70	0,606	0,7845	1,4536	0,5458
80	0,4729	0,8284	1,7317	0,5117
90	0,3534	0,8727	0,7360	0,5394
100	0,282	0,8975	1,8288	0,5139
110	0,2231	0,925	1,934	0,5352
120	0,184	0,9342	2,0181	0,533
130	0,155	0,4977	2,0439	0,5394
140	0,1286	0,9604	2,128	0,4309
150	0,1067	0,963	2,2798	0,5458
160	0,101	0,9667	2,281	0,5245
170	0,0896	0,9699	2,2833	0,5565
180	0,0769	0,792	2,5984	0,5352
190	0,0748	0,9762	2,5019	0,5267
200	0,0559	0,9831	2,4956	0,5501

Pelatihan model menggunakan data sebanyak 3738 data *training* dan 469 data validasi. Pelatihan menghabiskan waktu 1 hari 18 jam 12 menit 54 detik.

4.1.5 Melakukan pengujian model

Pengujian dilakukan menggunakan dengan 4 data yaitu, CK+, FER2013, KDEF, dan gabungan dari ketiga dataset. Pada percobaan pertama mencoba dengan mengubah optimasi yang digunakan. Terdapat 3 optimasi yang dicoba yaitu Adam, SGD, dan RMS. Semua dilakukan dengan menggunakan konfigurasi yang sama yaitu, *epoch* sebesar 200, *batch size* 64, dan dengan *learning rate* 0,0001.

Tabel 4.4 Perbandingan hasil dengan optimasi berbeda

Dataset	Akurasi		
	Adam	SGD	RMS
FER	0,52	0,19	0,48
KDEF	0,82	0,14	0,75
CK+	0,77	0,23	0,71

Gabungan	0,57	0,19	0,53

Dari Tabel 4.4. Nilai terbaik pada optimasi Adam. Pada data uji KDEF memiliki nilai paling tinggi hingga 0,81. Selanjutnya mencoba menguji dengan optimasi terbaik yaitu adam dengan perbedaan *epoch* saat melakukan pelatihan.

Tabel 4.5 Perbandingan hasil dengan epoch yang berbeda

Dataset	***Epoch***	
	200	**500**
FER	0,52	0,49
KDEF	0,81	0,77
CK+	0,77	0,71
Gabungan	0,57	0,54

4.2 Perbandingan hasil

Dalam pengujian model dapat dilihat akurasi tertinggi terdapat pada optimasi adam. Pengujian model dilakukan dengan empat data berbeda. Data tersebut yaitu data FER2013, CK+, KDEF, dan gabungan dari ketiga data tersebut. Pada perbandingan hasil ini akan dipilkan confussion matriks dari masing-masing dataset. Hasil uji ditampilkan dalam Confusion Matrix. Berikut baris code untuk pengujian dengan dataset gabungan,

```
    test_true = np.argmax(labels_test, axis=1)
    test_pred = np.argmax(model.predict(test_features), axis=1)
    print(metrics.confusion_matrix(test_true, test_pred,
labels=[0,1,2,3,4,5,6]))
    print(metrics.classification_report(test_true, test_pred,
labels=[0,1,2,3,4,5,6]))
```

Gambar 4.20 Kode untuk melakukan pengujian dengan data uji gabungan

Kode pada Gambar 4.14 fungsi *argmax*() digunakan untuk mendapatkabn indeks nilai tertinggi disepanjang sumbu. Test_true menyimpan isi dari variabel labels_test. Fungsi *predict()* digunakan untuk melakukan prediksi dengan model yang telah terlatih. Test_pred berisi hasil prediksi dari variabel test_features. Fungsi confusion_matrix digunakan untuk mengevaluasi keakuratan klasifikasi. Fungsi classification_*report()* untuk membangun laporan teks yang menunjukkan metrik klasifikasi utama. Pada fungsi tersebut terdapat recall, presisi, dan ukuran F.

Tabel 4.6 Confussion matrix dengan optimasi ADAM data uji gabungan

		Prediksi						
		0	1	2	3	4	5	6
Aktual	0	35	8	4	5	7	7	2
	1	12	31	3	3	4	8	7
	2	4	3	50	1	2	6	2
	3	8	13	4	21	4	10	8
	4	5	1	1	1	55	4	1
	5	16	10	2	5	6	27	2
	6	3	1	1	7	1	3	52

Tabel 4.7 Hasil pengujian dengan optimasi ADAM data uji gabungan

	Precision	Recall	F1-score	Support
0	0,42	0,51	0,46	68
1	0,46	0,46	0,46	68
2	0,77	0,74	0,75	68
3	0,49	0,31	0,38	68
4	0,7	0,81	0,78	68
5	0,42	0,4	0,41	68
6	0,7	0,76	0,73	68
Accuracy			0,57	476
Macro avg	0,57	0,57	0,56	476
Weighted avg	0,57	0,57	0,56	476

Tabel 4.8 Confussion matrix dengan optimasi ADAM data uji CK+

		Prediksi						
		0	1	2	3	4	5	6
Aktual	0	2	0	0	0	0	0	0
	1	1	3	0	0	0	1	0
	2	0	0	6	0	0	0	0
	3	0	0	0	2	0	1	0
	4	0	0	0	0	7	0	0
	5	1	2	0	0	0	0	0
	6	1	0	0	1	0	0	7

Tabel 4.9 Hasil pengujian dengan optimasi ADAM data uji CK+

	Precision	Recall	F1-score	Support
0	0,4	1	0,57	2

1	0,6	0,6	0,6	5
2	1	1	1	6
3	0,67	0,67	0,67	3
4	1	1	1	7
5	0	0	0	3
6	1	0,78	0,88	9
Accuracy			0,77	35
Macro avg	0,67	0,72	0,67	35
Weighted avg	0,79	0,77	0,77	35

Tabel 4.10 Confussion matrix dengan optimasi ADAM data uji FER2013

		Prediksi						
		0	1	2	3	4	5	6
Aktual	0	28	7	4	5	7	7	2
	1	10	22	3	3	4	7	7
	2	4	3	37	1	2	6	2
	3	7	13	4	14	4	8	8
	4	4	1	1	1	42	4	1
	5	14	6	1	5	6	24	2
	6	2	1	1	6	1	3	38

Tabel 4.11 Hasil pengujian dengan optimasi ADAM data FER2013

	Precision	Recall	F1-score	Support
0	0,41	0,47	0,43	60
1	0,42	0,39	0,4	56
2	0,73	0,67	0,7	55
3	0,4	0,24	0,3	58
4	0,64	0,72	0,7	54
5	0,41	0,41	0,41	58
6	0,63	0,73	0,68	52
Accuracy			0,52	393
Macro avg	0,52	0,53	0,52	393
Weighted avg	0,51	0,52	0,51	393

Tabel 4.12 Confussion matrix dengan optimasi ADAM data uji KDEF

		Prediksi						
		0	1	2	3	4	5	6
Aktual	0	6	1	0	0	0	0	0
	1	1	6	0	0	0	0	0
	2	0	0	7	0	0	0	0
	3	1	0	0	5	0	1	0
	4	1	0	0	0	6	0	0
	5	1	2	1	0	0	3	0
	6	0	0	0	0	0	0	7

Tabel 4.13 Hasil pengujian dengan optimasi ADAM data uji KDEF

	Precision	Recall	F1-score	Support
0	0,6	0,86	0,71	7
1	0,67	0,86	0,75	7
2	0,88	1	0,93	7
3	1	0,71	0,83	7
4	1	0,86	0,92	7
5	0,75	0,43	0,55	7
6	1	1	1	7
Accuracy			0,82	49
Macro avg	0,84	0,82	0,81	49
Weighted avg	0,84	0,82	0,81	49

Tabel 4.6 hingga Tabel 4.13 merupakan hasil menggunakan arsitektur yang tertera pada Bab III. Arsitektur ini menggunakan optimasi ADAM dengan *learning rate* 0,0001. Dari tabel 4.6 hingga Tabel 4.13 dapat diketahui akurasi paling besar terdapat pada dataset KDEF dan akurasi terkecil pada dataset FER2013. Emosi dengan akurasi tertinggi pada dataset gabungan yaitu emosi jijik. Emosi dengan akurasi tertinggi pada dataset FER yaitu emosi jijik dan untuk akurasi terendah yaiu emosi netral dan sedih. Emosi dengan akurasi tertinggi pada dataset CK+ yaitu emosi jijik, senang, dan terkejut. Emosi terendah pada dataset CK+ yaitu emosi sedih. Emosi dengan akurasi tertinggi pada dataset KDEF yaitu emosi takut, senang, dan terkejut. Emosi terendah pada dataset KDEF yaitu emosi netral. Dari keempat dataset jijik mendapatkan tingkat akurasi yang tinggi dan sedih mendapatkan akurasi yang rendah.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil analisis yang telah dilakukan, diperoleh beberapa kesimpulan sebagai berikut:

1. Model terbentuk dari hasil *training* dengan dengan menggunakan *batch size* 64 dengan pengujian 3 optimasi yang berbeda dan dengan epoch yang berbeda. Perbandingan dataset yang digunakan yaitu 80% training, 10% validasi, dan 10% test.
2. Optimasi yang terbaik yaitu dengan menggunakan optimasi Adam.
3. Hasil pengujian pada dataset KDEF memiliki nilai yang terbaik dibandingkan dengan dataset lainnya.
4. Emosi dengan tingkat akurasi tertinggi dimiliki oleh jijik dan emosi dengan akurasi terendah dimiliki oleh sedih.

5.2 Saran

Berdasarkan penelitian yang telah dilakukan, dapat diberikan beberapa saran sebagai berikut:

1. Membuat user interfaces menggunakan model yang telah terbuat.
2. Menggunakan dataset dengan warna yang RGB.
3. Menggunakan spesifikasi komputer processor yang tinggi, memiliki GPU yang bagus, dan RAM yang besar untuk mempercepat proses pelatihan model.

DAFTAR PUSTAKA

Abidin, Z. (2011). Pengembangan Sistem Pengenalan Ekspresi Wajah menggunakan Jaringan Syaraf Tiruan Backpropagation (Studi Kasus pada Database MUG). *Jurnal Matematika Murni Dan Terapan*, *5*(1), 21–30.

Achmad, Y., Wihandika, R. C., & Dewi, C. (2019). Klasifikasi Emosi Berdasarkan Ciri Wajah Menggunakan Convolutional Neural Network. *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer*, *3*(11), 10595–10604.

Alexander, S. (2013). *Program Aplikasi Pengenalan Ekspresi Wajah Secara Real Time Dengan Metode Back Propagation Dan Wavelet Haar*.

Amynarto, N., Sari, Y. A., & Cahyawihandika, R. (2018). Pengenalan Emosi Berdasarkan Ekspresi Mikro Menggunakan Metode Local Binary Pattern. *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer (J-PTIIK) Universitas Brawijaya*, *2*(10), 3230–3238.

Astuti, D. L. Z. (2019). *Klasifikasi ekspresi wajah menggunakan metode principal component analysis (pca) dan convolutional neural network (cnn)*. 1–80. https://repository.unsri.ac.id/6479/

Çarıkçı, M. üg., & Özen, F. (2012). A Face Recognition System Based on Eigenfaces Method. *Procedia Technology*, *1*, 118–123. https://doi.org/10.1016/j.protcy.2012.02.023

Carrier, P.-L., & Courville, A. (2013). *Challenges in Representation Learning: Facial Expression Recognition Challenge*. https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data

Eka Putra, W. S. (2016). Klasifikasi Citra Menggunakan Convolutional Neural Network (CNN) pada Caltech 101. *Jurnal Teknik ITS*, *5*(1). https://doi.org/10.12962/j23373539.v5i1.15696

Ekman, P. (2016). *Nonverbal Messages: Cracking the Code: My LIfe's Pursuit*. Paul Ekman Group.

Fabri, M. (2004). *Mediating the Expression of Emotion in Educational CVEs: An Experimental Study*. https://www.researchgate.net/figure/The-six-universal-emotions-and-neutral-expression_fig1_240113130

Goleman, D. (2002). *Kecerdasan Emosional*. PT Gamedia Pustaka Utama.

Hakim, D. M., & Rainarli, E. (2019). Convolutional Neural Network untuk Pengenalan Citra Notasi Musik. *Techno.Com*, *18*(3), 214–226. https://doi.org/10.33633/tc.v18i3.2387

Hendriyana, & Maulana, Y. H. (2020). Identifikasi Jenis Kayu menggunakan Convolutional Neural Network. *Jurnal RESTI (Rekayasa Sistem Dan Teknologi Informasi)*, *4*(1), 70–76.

Kusumawati, T. I. (2016). Komunikasi Verbal Dan Nonverbal. *Jurnal Pendidikan Dan Konseling*, *6*(2), 83–98.

L.Pt. Purnamaningsih, Ni Kt. Suarni, K. S. (2019). *Identifikasi Emosi Melalui Pendeteksian Karakteristik Ekspresi Wajah (Face Expression) Dalam Rangka Mengentaskan Masalah Siswa Melalui Konseling Individual*. *44*(12), 2–8. https://doi.org/10. 19540 /j. cnki. cjcmm. 20190128. 002

Nabuasa, Y. N. (2019). Pengolahan Citra Digital Perbandingan Metode Histogram Equalization Dan Spesification Pada Citra Abu-Abu. *J-Icon*, *7*(1), 87–95.

Narkhede, S. (2018). *Understanding Confusion Matrix*. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

Nugroho, P. A., Fenriana, I., Arijanto, R., & Kom, M. (2020). *IMPLEMENTASI DEEP LEARNING MENGGUNAKAN CONVOLUTIONAL NEURAL NETWORK (CNN) PADA EKSPRESI MANUSIA*. *1*.

Oliver, M. M., & Alcover, E. A. (2020). UIBVFed: Virtual facial expression dataset. *PLoS ONE*, *15*(4), 1–10. https://doi.org/10.1371/journal.pone.0231266

Planalp, S. (2015). How Important Is Emotion in Everyday Interaction? *Communicating Emotion*, 9–38. https://doi.org/10.1017/cbo9781316257012.003

Rahman, M. F., Alamsah, D., Darmawidjadja, M. I., & Nurma, I. (2017). Klasifikasi Untuk Diagnosa Diabetes Menggunakan Metode Bayesian Regularization Neural Network (RBNN). *Jurnal Informatika*, *11*(1), 36. https://doi.org/10.26555/jifo.v11i1.a5452

Ramdhani, B., Djamal, E. C., & Ilyas, R. (2018). Convolutional Neural Networks Models for Facial Expression Recognition. *2018 International Symposium on Advanced Intelligent Informatics (SAIN)*, 96–101.

Sang, D. V., Van Dat, N., & Thuan, D. P. (2017). Facial expression recognition using deep convolutional neural networks. *Proceedings - 2017 9th International Conference on Knowledge and Systems Engineering, KSE 2017*, *2017-Janua*, 130–135. https://doi.org/10.1109/KSE.2017.8119447

Santoso, A., & Ariyanto, G. (2018). Implementasi Deep Learning Berbasis Keras Untuk Pengenalan Wajah. *Emitor: Jurnal Teknik Elektro*, *18*(01), 15–21. https://doi.org/10.23917/emitor.v18i01.6235

Septian, R., Saputra, D. I., & Sambasri, S. (n.d.). *Klasifikasi Emosi Menggunakan*

Convolutional Neural Networks. *November 2019*, 53–62.

Stazio, A., Victores, J. G., Estevez, D., & Balaguer, C. (2019). *A Study on Machine Vision Techniques for the Inspection of Health Personnels' Protective Suits for the Treatment of Patients in Extreme Isolation*.

Tri Anindia Putra, I. N. (2015). Perancangan dan Pengembangan Sistem Absensi Realtime Melalui Metode Pengenalan Wajah. *JST (Jurnal Sains Dan Teknologi)*, *3*(2), 450–467. https://doi.org/10.23887/jst-undiksha.v3i2.4480

Walters, A. (2019). *Max Pooling*. https://austingwalters.com/convolutional-neural-networks-cnn-to-classify-sentences/max-pooling/

Zufar, M., & Setiyono, B. (2016). Convolutional Neural Networks Untuk Pengenalan Wajah Secara Real-time. *Jurnal Sains Dan Seni ITS*, *5*(2), 128862.

LAMPIRAN

```
Train on 3787 samples, validate on 469 samples
Epoch 1/200
3787/3787 [==============================] - 734s 194ms/step - loss: 2.2896
- accuracy: 0.1584 - val_loss: 1.9662 - val_accuracy: 0.1429
Epoch 2/200
C:\ProgramData\Anaconda3\envs\FazaEnv\lib\site-packages\keras\callbacks\cal
lbacks.py:707: RuntimeWarning: Can save best model only with val_acc availa
ble, skipping.
  'skipping.' % (self.monitor), RuntimeWarning)
3787/3787 [==============================] - 784s 207ms/step - loss: 2.1433
- accuracy: 0.1888 - val_loss: 2.0459 - val_accuracy: 0.1429
Epoch 3/200
3787/3787 [==============================] - 772s 204ms/step - loss: 2.0921
- accuracy: 0.1936 - val_loss: 2.1602 - val_accuracy: 0.1429
Epoch 4/200
3787/3787 [==============================] - 774s 204ms/step - loss: 2.0541
- accuracy: 0.2054 - val_loss: 2.1343 - val_accuracy: 0.1429
Epoch 5/200
3787/3787 [==============================] - 769s 203ms/step - loss: 2.0227
- accuracy: 0.2168 - val_loss: 2.1361 - val_accuracy: 0.1429
Epoch 6/200
3787/37870 [==============================] - 775s 205ms/step - loss: 1.978
4 - accuracy: 0.2358 - val_loss: 2.1949 - val_accuracy: 0.1429
Epoch 7/200
3787/3787 [==============================] - 753s 199ms/step - loss: 1.9476
- accuracy: 0.2564 - val_loss: 2.1886 - val_accuracy: 0.1429
Epoch 8/200
3787/3787 [==============================] - 752s 199ms/step - loss: 1.8827
- accuracy: 0.2807 - val_loss: 2.1407 - val_accuracy: 0.1429
Epoch 9/200
3787/3787 [==============================] - 752s 198ms/step - loss: 1.8603
- accuracy: 0.2749 - val_loss: 2.0912 - val_accuracy: 0.1407
Epoch 10/200
3787/3787 [==============================] - 751s 198ms/step - loss: 1.8246
- accuracy: 0.2973 - val_loss: 2.1075 - val_accuracy: 0.1535
Epoch 11/200
3787/3787 [==============================] - 757s 200ms/step - loss: 1.7955
- accuracy: 0.3047 - val_loss: 1.9880 - val_accuracy: 0.2154
Epoch 12/200
3787/3787 [==============================] - 756s 200ms/step - loss: 1.7555
- accuracy: 0.3282 - val_loss: 1.8917 - val_accuracy: 0.2836
Epoch 13/200
3787/3787 [==============================] - 758s 200ms/step - loss: 1.7414
- accuracy: 0.3330 - val_loss: 1.8546 - val_accuracy: 0.2921
```

```
Epoch 14/200
3787/3787 [==============================] - 759s 200ms/step - loss: 1.7144
- accuracy: 0.3454 - val_loss: 1.8179 - val_accuracy: 0.3156
Epoch 15/200
3787/3787 [==============================] - 756s 200ms/step - loss: 1.6780
- accuracy: 0.3491 - val_loss: 1.7864 - val_accuracy: 0.3369
Epoch 16/200
3787/3787 [==============================] - 755s 199ms/step - loss: 1.6582
- accuracy: 0.3692 - val_loss: 1.7963 - val_accuracy: 0.3582
Epoch 17/200
3787/3787 [==============================] - 710s 188ms/step - loss: 1.6584
- accuracy: 0.3713 - val_loss: 1.7764 - val_accuracy: 0.3731
Epoch 18/200
3787/3787 [==============================] - 789s 208ms/step - loss: 1.6084
- accuracy: 0.3810 - val_loss: 1.7894 - val_accuracy: 0.3753
Epoch 19/200
3787/3787 [==============================] - 780s 206ms/step - loss: 1.5837
- accuracy: 0.3874 - val_loss: 1.7214 - val_accuracy: 0.3753
Epoch 20/200
3787/3787 [==============================] - 785s 207ms/step - loss: 1.5425
- accuracy: 0.4164 - val_loss: 1.7153 - val_accuracy: 0.3731
Epoch 21/200
3787/3787 [==============================] - 781s 206ms/step - loss: 1.5319
- accuracy: 0.4175 - val_loss: 1.7441 - val_accuracy: 0.3731
Epoch 22/200
3787/3787 [==============================] - 761s 201ms/step - loss: 1.5023
- accuracy: 0.4225 - val_loss: 1.6400 - val_accuracy: 0.3881
Epoch 23/200
3787/3787 [==============================] - 757s 200ms/step - loss: 1.5003
- accuracy: 0.4273 - val_loss: 1.6634 - val_accuracy: 0.4094
Epoch 24/200
3787/3787 [==============================] - 756s 200ms/step - loss: 1.4640
- accuracy: 0.4386 - val_loss: 1.7115 - val_accuracy: 0.3923
Epoch 25/200
3787/3787 [==============================] - 755s 199ms/step - loss: 1.4242
- accuracy: 0.4566 - val_loss: 1.5450 - val_accuracy: 0.4158
Epoch 26/200
3787/3787 [==============================] - 756s 200ms/step - loss: 1.3992
- accuracy: 0.4682 - val_loss: 1.5932 - val_accuracy: 0.4350
Epoch 27/200
3787/3787 [==============================] - 761s 201ms/step - loss: 1.4109
- accuracy: 0.4721 - val_loss: 1.5955 - val_accuracy: 0.4414
Epoch 28/200
3787/3787 [==============================] - 757s 200ms/step - loss: 1.3819
- accuracy: 0.4806 - val_loss: 1.6067 - val_accuracy: 0.4350
Epoch 29/200
```

```
3787/3787 [==============================] - 760s 201ms/step - loss: 1.3545
- accuracy: 0.4840 - val_loss: 1.6032 - val_accuracy: 0.4392
Epoch 30/200
3787/3787 [==============================] - 760s 201ms/step - loss: 1.3293
- accuracy: 0.4935 - val_loss: 1.5342 - val_accuracy: 0.4563
Epoch 31/200
3787/3787 [==============================] - 758s 200ms/step - loss: 1.3255
- accuracy: 0.4978 - val_loss: 1.5686 - val_accuracy: 0.4520
Epoch 32/200
3787/3787 [==============================] - 757s 200ms/step - loss: 1.3022
- accuracy: 0.5207 - val_loss: 1.5487 - val_accuracy: 0.4456
Epoch 33/200
3787/3787 [==============================] - 755s 199ms/step - loss: 1.2623
- accuracy: 0.5186 - val_loss: 1.5744 - val_accuracy: 0.4435
Epoch 34/200
3787/3787 [==============================] - 757s 200ms/step - loss: 1.2367
- accuracy: 0.5292 - val_loss: 1.4816 - val_accuracy: 0.4733
Epoch 35/200
3787/3787 [==============================] - 753s 199ms/step - loss: 1.2383
- accuracy: 0.5400 - val_loss: 1.4429 - val_accuracy: 0.4691
Epoch 36/200
3787/3787 [==============================] - 756s 200ms/step - loss: 1.1984
- accuracy: 0.5442 - val_loss: 1.5200 - val_accuracy: 0.4563
Epoch 37/200
3787/3787 [==============================] - 752s 199ms/step - loss: 1.1949
- accuracy: 0.5527 - val_loss: 1.4759 - val_accuracy: 0.4755
Epoch 38/200
3787/3787 [==============================] - 760s 201ms/step - loss: 1.1712
- accuracy: 0.5548 - val_loss: 1.4763 - val_accuracy: 0.4776
Epoch 39/200
3787/3787 [==============================] - 875s 231ms/step - loss: 1.1498
- accuracy: 0.5741 - val_loss: 1.5543 - val_accuracy: 0.4670
Epoch 40/200
3787/3787 [==============================] - 797s 211ms/step - loss: 1.1363
- accuracy: 0.5749 - val_loss: 1.4971 - val_accuracy: 0.4797
Epoch 41/200
3787/3787 [==============================] - 802s 212ms/step - loss: 1.0992
- accuracy: 0.5889 - val_loss: 1.5011 - val_accuracy: 0.4691
Epoch 42/200
3787/3787 [==============================] - 818s 216ms/step - loss: 1.0878
- accuracy: 0.5915 - val_loss: 1.5101 - val_accuracy: 0.4755
Epoch 43/200
3787/3787 [==============================] - 785s 207ms/step - loss: 1.0850
- accuracy: 0.5941 - val_loss: 1.4730 - val_accuracy: 0.4840
Epoch 44/200
3787/3787 [==============================] - 782s 206ms/step - loss: 1.0658
- accuracy: 0.5963 - val_loss: 1.4570 - val_accuracy: 0.4968
```

```
Epoch 45/200
3787/3787 [==============================] - 770s 203ms/step - loss: 1.0440
- accuracy: 0.6044 - val_loss: 1.4618 - val_accuracy: 0.4904
Epoch 46/200
3787/3787 [==============================] - 757s 200ms/step - loss: 1.0173
- accuracy: 0.6150 - val_loss: 1.4447 - val_accuracy: 0.4925
Epoch 47/200
3787/3787 [==============================] - 753s 199ms/step - loss: 1.0014
- accuracy: 0.6282 - val_loss: 1.4739 - val_accuracy: 0.4883
Epoch 48/200
3787/3787 [==============================] - 839s 222ms/step - loss: 0.9734
- accuracy: 0.6340 - val_loss: 1.4224 - val_accuracy: 0.5096
Epoch 49/200
3787/3787 [==============================] - 777s 205ms/step - loss: 0.9712
- accuracy: 0.6348 - val_loss: 1.4644 - val_accuracy: 0.4925
Epoch 50/200
3787/3787 [==============================] - 777s 205ms/step - loss: 0.9505
- accuracy: 0.6364 - val_loss: 1.4613 - val_accuracy: 0.4925
Epoch 51/200
3787/3787 [==============================] - 778s 205ms/step - loss: 0.9246
- accuracy: 0.6506 - val_loss: 1.4779 - val_accuracy: 0.5224
Epoch 52/200
3787/3787 [==============================] - 778s 205ms/step - loss: 0.9192
- accuracy: 0.6543 - val_loss: 1.4989 - val_accuracy: 0.4925
Epoch 53/200
3787/3787 [==============================] - 777s 205ms/step - loss: 0.8786
- accuracy: 0.6763 - val_loss: 1.4958 - val_accuracy: 0.5032
Epoch 54/200
3787/3787 [==============================] - 781s 206ms/step - loss: 0.8602
- accuracy: 0.6889 - val_loss: 1.5480 - val_accuracy: 0.4947
Epoch 55/200
3787/3787 [==============================] - 780s 206ms/step - loss: 0.8376
- accuracy: 0.6929 - val_loss: 1.4422 - val_accuracy: 0.5075
Epoch 56/200
3787/3787 [==============================] - 779s 206ms/step - loss: 0.8282
- accuracy: 0.7050 - val_loss: 1.4172 - val_accuracy: 0.5330
Epoch 57/200
3787/3787 [==============================] - 777s 205ms/step - loss: 0.8107
- accuracy: 0.7024 - val_loss: 1.4551 - val_accuracy: 0.5139
Epoch 58/200
3787/3787 [==============================] - 777s 205ms/step - loss: 0.8144
- accuracy: 0.6987 - val_loss: 1.4284 - val_accuracy: 0.5309
Epoch 59/200
3787/3787 [==============================] - 783s 207ms/step - loss: 0.7873
- accuracy: 0.7085 - val_loss: 1.4451 - val_accuracy: 0.5416
Epoch 60/200
```

```
3787/3787 [==============================] - 781s 206ms/step - loss: 0.7659
- accuracy: 0.7193 - val_loss: 1.4654 - val_accuracy: 0.5032
Epoch 61/200
3787/3787 [==============================] - 777s 205ms/step - loss: 0.7521
- accuracy: 0.7262 - val_loss: 1.4519 - val_accuracy: 0.5245
Epoch 62/200
3787/3787 [==============================] - 775s 205ms/step - loss: 0.7407
- accuracy: 0.7312 - val_loss: 1.4890 - val_accuracy: 0.5267
Epoch 63/200
3787/3787 [==============================] - 782s 207ms/step - loss: 0.7037
- accuracy: 0.7478 - val_loss: 1.4455 - val_accuracy: 0.5267
Epoch 64/200
3787/3787 [==============================] - 780s 206ms/step - loss: 0.6875
- accuracy: 0.7457 - val_loss: 1.4409 - val_accuracy: 0.5330
Epoch 65/200
3787/3787 [==============================] - 786s 208ms/step - loss: 0.7010
- accuracy: 0.7462 - val_loss: 1.4725 - val_accuracy: 0.5245
Epoch 66/200
3787/3787 [==============================] - 778s 205ms/step - loss: 0.6696
- accuracy: 0.7576 - val_loss: 1.4562 - val_accuracy: 0.5522
Epoch 67/200
3787/3787 [==============================] - 783s 207ms/step - loss: 0.6677
- accuracy: 0.7507 - val_loss: 1.4859 - val_accuracy: 0.5394
Epoch 68/200
3787/3787 [==============================] - 773s 204ms/step - loss: 0.6469
- accuracy: 0.7626 - val_loss: 1.5196 - val_accuracy: 0.5224
Epoch 69/200
3787/3787 [==============================] - 758s 200ms/step - loss: 0.6313
- accuracy: 0.7671 - val_loss: 1.5622 - val_accuracy: 0.5224
Epoch 70/200
3787/3787 [==============================] - 759s 200ms/step - loss: 0.6060
- accuracy: 0.7845 - val_loss: 1.4536 - val_accuracy: 0.5458
Epoch 71/200
3787/3787 [==============================] - 756s 200ms/step - loss: 0.6088
- accuracy: 0.7766 - val_loss: 1.4658 - val_accuracy: 0.5330
Epoch 72/200
3787/3787 [==============================] - 757s 200ms/step - loss: 0.5744
- accuracy: 0.7940 - val_loss: 1.5519 - val_accuracy: 0.5330
Epoch 73/200
3787/3787 [==============================] - 759s 200ms/step - loss: 0.5416
- accuracy: 0.8099 - val_loss: 1.5833 - val_accuracy: 0.5394
Epoch 74/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.5485
- accuracy: 0.8030 - val_loss: 1.5431 - val_accuracy: 0.5629
Epoch 75/200
3787/3787 [==============================] - 755s 199ms/step - loss: 0.5196
- accuracy: 0.8191 - val_loss: 1.5764 - val_accuracy: 0.5330
```

```
Epoch 76/200
3787/3787 [==============================] - 760s 201ms/step - loss: 0.5118
- accuracy: 0.8257 - val_loss: 1.5436 - val_accuracy: 0.5352
Epoch 77/200
3787/3787 [==============================] - 756s 200ms/step - loss: 0.5039
- accuracy: 0.8170 - val_loss: 1.5198 - val_accuracy: 0.5501
Epoch 78/200
3787/3787 [==============================] - 754s 199ms/step - loss: 0.4905
- accuracy: 0.8215 - val_loss: 1.6616 - val_accuracy: 0.5330
Epoch 79/200
3787/3787 [==============================] - 755s 199ms/step - loss: 0.4918
- accuracy: 0.8286 - val_loss: 1.5466 - val_accuracy: 0.5458
Epoch 80/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.4729
- accuracy: 0.8284 - val_loss: 1.7317 - val_accuracy: 0.5117
Epoch 81/200
3787/3787 [==============================] - 759s 200ms/step - loss: 0.4742
- accuracy: 0.8336 - val_loss: 1.6250 - val_accuracy: 0.5458
Epoch 82/200
3787/3787 [==============================] - 748s 198ms/step - loss: 0.4422
- accuracy: 0.8418 - val_loss: 1.6969 - val_accuracy: 0.5437
Epoch 83/200
3787/3787 [==============================] - 758s 200ms/step - loss: 0.4258
- accuracy: 0.8537 - val_loss: 1.6090 - val_accuracy: 0.5330
Epoch 84/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.4227
- accuracy: 0.8505 - val_loss: 1.5895 - val_accuracy: 0.5501
Epoch 85/200
3787/3787 [==============================] - 755s 199ms/step - loss: 0.4172
- accuracy: 0.8521 - val_loss: 1.6982 - val_accuracy: 0.5330
Epoch 86/200
3787/3787 [==============================] - 752s 199ms/step - loss: 0.3973
- accuracy: 0.8643 - val_loss: 1.6464 - val_accuracy: 0.5160
Epoch 87/200
3787/3787 [==============================] - 757s 200ms/step - loss: 0.3938
- accuracy: 0.8669 - val_loss: 1.6529 - val_accuracy: 0.5416
Epoch 88/200
3787/3787 [==============================] - 751s 198ms/step - loss: 0.3859
- accuracy: 0.8645 - val_loss: 1.7019 - val_accuracy: 0.5288
Epoch 89/200
3787/3787 [==============================] - 757s 200ms/step - loss: 0.3820
- accuracy: 0.8688 - val_loss: 1.7222 - val_accuracy: 0.5373
Epoch 90/200
3787/3787 [==============================] - 758s 200ms/step - loss: 0.3534
- accuracy: 0.8727 - val_loss: 1.7360 - val_accuracy: 0.5394
Epoch 91/200
```

```
3787/3787 [==============================] - 755s 199ms/step - loss: 0.3559
- accuracy: 0.8775 - val_loss: 1.6672 - val_accuracy: 0.5309
Epoch 92/200
3787/3787 [==============================] - 757s 200ms/step - loss: 0.3503
- accuracy: 0.8735 - val_loss: 1.6699 - val_accuracy: 0.5416
Epoch 93/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.3413
- accuracy: 0.8835 - val_loss: 1.7842 - val_accuracy: 0.5203
Epoch 94/200
3787/3787 [==============================] - 752s 199ms/step - loss: 0.3244
- accuracy: 0.8915 - val_loss: 1.6735 - val_accuracy: 0.5309
Epoch 95/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.3333
- accuracy: 0.8835 - val_loss: 1.6711 - val_accuracy: 0.5245
Epoch 96/200
3787/3787 [==============================] - 751s 198ms/step - loss: 0.3079
- accuracy: 0.8878 - val_loss: 1.7793 - val_accuracy: 0.5203
Epoch 97/200
3787/3787 [==============================] - 757s 200ms/step - loss: 0.3077
- accuracy: 0.8962 - val_loss: 1.7627 - val_accuracy: 0.5288
Epoch 98/200
3787/3787 [==============================] - 752s 199ms/step - loss: 0.2983
- accuracy: 0.8981 - val_loss: 1.7711 - val_accuracy: 0.5437
Epoch 99/200
3787/3787 [==============================] - 754s 199ms/step - loss: 0.2932
- accuracy: 0.8899 - val_loss: 1.8487 - val_accuracy: 0.5203
Epoch 100/200
3787/3787 [==============================] - 754s 199ms/step - loss: 0.2820
- accuracy: 0.8975 - val_loss: 1.8288 - val_accuracy: 0.5139
Epoch 101/200
3787/3787 [==============================] - 778s 205ms/step - loss: 0.2603
- accuracy: 0.9105 - val_loss: 1.7523 - val_accuracy: 0.5480
Epoch 102/200
3787/3787 [==============================] - 785s 207ms/step - loss: 0.2766
- accuracy: 0.9015 - val_loss: 1.7911 - val_accuracy: 0.5522
Epoch 103/200
3787/3787 [==============================] - 779s 206ms/step - loss: 0.2723
- accuracy: 0.9084 - val_loss: 1.8424 - val_accuracy: 0.5288
Epoch 104/200
3787/3787 [==============================] - 778s 205ms/step - loss: 0.2546
- accuracy: 0.9171 - val_loss: 1.8165 - val_accuracy: 0.5394
Epoch 105/200
3787/3787 [==============================] - 776s 205ms/step - loss: 0.2387
- accuracy: 0.9192 - val_loss: 1.8544 - val_accuracy: 0.5480
Epoch 106/200
3787/3787 [==============================] - 770s 203ms/step - loss: 0.2429
- accuracy: 0.9179 - val_loss: 1.8478 - val_accuracy: 0.5458
```

```
Epoch 107/200
3787/3787 [==============================] - 750s 198ms/step - loss: 0.2346
- accuracy: 0.9189 - val_loss: 1.8383 - val_accuracy: 0.5501
Epoch 108/200
3787/3787 [==============================] - 752s 199ms/step - loss: 0.2595
- accuracy: 0.9044 - val_loss: 2.0182 - val_accuracy: 0.5330
Epoch 109/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.2303
- accuracy: 0.9245 - val_loss: 1.8462 - val_accuracy: 0.5373
Epoch 110/200
3787/3787 [==============================] - 752s 199ms/step - loss: 0.2231
- accuracy: 0.9250 - val_loss: 1.9340 - val_accuracy: 0.5352
Epoch 111/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.2223
- accuracy: 0.9306 - val_loss: 1.9235 - val_accuracy: 0.5416
Epoch 112/200
3787/3787 [==============================] - 756s 200ms/step - loss: 0.2363
- accuracy: 0.9195 - val_loss: 1.8888 - val_accuracy: 0.5437
Epoch 113/200
3787/3787 [==============================] - 751s 198ms/step - loss: 0.2121
- accuracy: 0.9276 - val_loss: 1.9617 - val_accuracy: 0.5373
Epoch 114/200
3787/3787 [==============================] - 754s 199ms/step - loss: 0.1891
- accuracy: 0.9395 - val_loss: 1.9276 - val_accuracy: 0.5352
Epoch 115/200
3787/3787 [==============================] - 756s 200ms/step - loss: 0.2083
- accuracy: 0.9237 - val_loss: 1.8880 - val_accuracy: 0.5437
Epoch 116/200
3787/3787 [==============================] - 757s 200ms/step - loss: 0.1971
- accuracy: 0.9382 - val_loss: 1.9482 - val_accuracy: 0.5522
Epoch 117/200
3787/3787 [==============================] - 761s 201ms/step - loss: 0.2234
- accuracy: 0.9240 - val_loss: 1.9355 - val_accuracy: 0.5501
Epoch 118/200
3787/3787 [==============================] - 755s 199ms/step - loss: 0.2026
- accuracy: 0.9250 - val_loss: 2.0500 - val_accuracy: 0.5224
Epoch 119/200
3787/3787 [==============================] - 754s 199ms/step - loss: 0.1797
- accuracy: 0.9353 - val_loss: 2.0484 - val_accuracy: 0.5437
Epoch 120/200
3787/3787 [==============================] - 754s 199ms/step - loss: 0.1840
- accuracy: 0.9342 - val_loss: 2.0181 - val_accuracy: 0.5330
Epoch 121/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.1678
- accuracy: 0.9453 - val_loss: 2.0211 - val_accuracy: 0.5373
Epoch 122/200
```

```
3787/3787 [==============================] - 753s 199ms/step - loss: 0.1825
- accuracy: 0.9366 - val_loss: 2.1023 - val_accuracy: 0.5330
Epoch 123/200
3787/3787 [==============================] - 754s 199ms/step - loss: 0.1657
- accuracy: 0.9443 - val_loss: 2.0184 - val_accuracy: 0.5394
Epoch 124/200
3787/3787 [==============================] - 752s 198ms/step - loss: 0.1662
- accuracy: 0.9422 - val_loss: 2.0460 - val_accuracy: 0.5352
Epoch 125/200
3787/3787 [==============================] - 754s 199ms/step - loss: 0.1471
- accuracy: 0.9535 - val_loss: 2.0706 - val_accuracy: 0.5267
Epoch 126/200
3787/3787 [==============================] - 752s 199ms/step - loss: 0.1668
- accuracy: 0.9472 - val_loss: 2.1143 - val_accuracy: 0.5117
Epoch 127/200
3787/3787 [==============================] - 755s 199ms/step - loss: 0.1565
- accuracy: 0.9459 - val_loss: 2.0667 - val_accuracy: 0.5160
Epoch 128/200
3787/3787 [==============================] - 755s 199ms/step - loss: 0.1604
- accuracy: 0.9453 - val_loss: 2.1103 - val_accuracy: 0.5224
Epoch 129/200
3787/3787 [==============================] - 755s 199ms/step - loss: 0.1590
- accuracy: 0.9435 - val_loss: 2.1401 - val_accuracy: 0.5288
Epoch 130/200
3787/3787 [==============================] - 765s 202ms/step - loss: 0.1550
- accuracy: 0.9477 - val_loss: 2.0439 - val_accuracy: 0.5394
Epoch 131/200
3787/3787 [==============================] - 761s 201ms/step - loss: 0.1436
- accuracy: 0.9538 - val_loss: 2.0734 - val_accuracy: 0.5394
Epoch 132/200
3787/3787 [==============================] - 754s 199ms/step - loss: 0.1382
- accuracy: 0.9530 - val_loss: 2.1789 - val_accuracy: 0.5117
Epoch 133/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.1373
- accuracy: 0.9548 - val_loss: 2.0800 - val_accuracy: 0.5437
Epoch 134/200
3787/3787 [==============================] - 754s 199ms/step - loss: 0.1491
- accuracy: 0.9480 - val_loss: 2.1294 - val_accuracy: 0.5544
Epoch 135/200
3787/3787 [==============================] - 756s 200ms/step - loss: 0.1384
- accuracy: 0.9527 - val_loss: 2.1740 - val_accuracy: 0.5267
Epoch 136/200
3787/3787 [==============================] - 754s 199ms/step - loss: 0.1312
- accuracy: 0.9530 - val_loss: 2.2476 - val_accuracy: 0.5203
Epoch 137/200
3787/3787 [==============================] - 761s 201ms/step - loss: 0.1516
- accuracy: 0.9472 - val_loss: 2.0762 - val_accuracy: 0.5437
```

```
Epoch 138/200
3787/3787 [==============================] - 757s 200ms/step - loss: 0.1236
- accuracy: 0.9588 - val_loss: 2.0550 - val_accuracy: 0.5416
Epoch 139/200
3787/3787 [==============================] - 757s 200ms/step - loss: 0.1366
- accuracy: 0.9567 - val_loss: 2.2012 - val_accuracy: 0.5352
Epoch 140/200
3787/3787 [==============================] - 755s 199ms/step - loss: 0.1286
- accuracy: 0.9604 - val_loss: 2.1280 - val_accuracy: 0.5309
Epoch 141/200
3787/3787 [==============================] - 752s 199ms/step - loss: 0.1179
- accuracy: 0.9583 - val_loss: 2.1091 - val_accuracy: 0.5352
Epoch 142/200
3787/3787 [==============================] - 755s 199ms/step - loss: 0.1144
- accuracy: 0.9591 - val_loss: 2.2090 - val_accuracy: 0.5203
Epoch 143/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.1152
- accuracy: 0.9636 - val_loss: 2.3433 - val_accuracy: 0.5075
Epoch 144/200
3787/3787 [==============================] - 751s 198ms/step - loss: 0.1239
- accuracy: 0.9546 - val_loss: 2.1814 - val_accuracy: 0.5394
Epoch 145/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.1174
- accuracy: 0.9593 - val_loss: 2.1552 - val_accuracy: 0.5330
Epoch 146/200
3787/3787 [==============================] - 751s 198ms/step - loss: 0.1096
- accuracy: 0.9625 - val_loss: 2.2546 - val_accuracy: 0.5160
Epoch 147/200
3787/3787 [==============================] - 756s 200ms/step - loss: 0.1087
- accuracy: 0.9638 - val_loss: 2.1304 - val_accuracy: 0.5522
Epoch 148/200
3787/3787 [==============================] - 752s 199ms/step - loss: 0.1079
- accuracy: 0.9644 - val_loss: 2.3493 - val_accuracy: 0.5160
Epoch 149/200
3787/3787 [==============================] - 756s 200ms/step - loss: 0.1045
- accuracy: 0.9638 - val_loss: 2.2765 - val_accuracy: 0.5032
Epoch 150/200
3787/3787 [==============================] - 754s 199ms/step - loss: 0.1067
- accuracy: 0.9630 - val_loss: 2.2798 - val_accuracy: 0.5458
Epoch 151/200
3787/3787 [==============================] - 756s 200ms/step - loss: 0.1061
- accuracy: 0.9638 - val_loss: 2.2584 - val_accuracy: 0.5394
Epoch 152/200
3787/3787 [==============================] - 756s 200ms/step - loss: 0.1107
- accuracy: 0.9633 - val_loss: 2.1861 - val_accuracy: 0.5394
Epoch 153/200
```

```
3787/3787 [==============================] - 780s 206ms/step - loss: 0.1065
- accuracy: 0.9638 - val_loss: 2.3244 - val_accuracy: 0.5352
Epoch 154/200
3787/3787 [==============================] - 781s 206ms/step - loss: 0.1015
- accuracy: 0.9675 - val_loss: 2.2143 - val_accuracy: 0.5416
Epoch 155/200
3787/3787 [==============================] - 779s 206ms/step - loss: 0.1118
- accuracy: 0.9620 - val_loss: 2.2980 - val_accuracy: 0.5330
Epoch 156/200
3787/3787 [==============================] - 783s 207ms/step - loss: 0.1018
- accuracy: 0.9667 - val_loss: 2.3259 - val_accuracy: 0.5267
Epoch 157/200
3787/3787 [==============================] - 777s 205ms/step - loss: 0.1054
- accuracy: 0.9628 - val_loss: 2.4265 - val_accuracy: 0.5203
Epoch 158/200
3787/3787 [==============================] - 778s 205ms/step - loss: 0.0954
- accuracy: 0.9694 - val_loss: 2.5277 - val_accuracy: 0.5139
Epoch 159/200
3787/3787 [==============================] - 756s 200ms/step - loss: 0.0958
- accuracy: 0.9680 - val_loss: 2.4026 - val_accuracy: 0.5394
Epoch 160/200
3787/3787 [==============================] - 757s 200ms/step - loss: 0.1010
- accuracy: 0.9667 - val_loss: 2.2810 - val_accuracy: 0.5245
Epoch 161/200
3787/3787 [==============================] - 756s 200ms/step - loss: 0.0963
- accuracy: 0.9678 - val_loss: 2.2374 - val_accuracy: 0.5394
Epoch 162/200
3787/3787 [==============================] - 755s 199ms/step - loss: 0.0841
- accuracy: 0.9725 - val_loss: 2.2324 - val_accuracy: 0.5458
Epoch 163/200
3787/3787 [==============================] - 756s 200ms/step - loss: 0.1042
- accuracy: 0.9607 - val_loss: 2.2511 - val_accuracy: 0.5267
Epoch 164/200
3787/3787 [==============================] - 756s 200ms/step - loss: 0.0889
- accuracy: 0.9702 - val_loss: 2.2969 - val_accuracy: 0.5458
Epoch 165/200
3787/3787 [==============================] - 756s 200ms/step - loss: 0.0894
- accuracy: 0.9715 - val_loss: 2.2521 - val_accuracy: 0.5394
Epoch 166/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.1010
- accuracy: 0.9636 - val_loss: 2.3703 - val_accuracy: 0.5629
Epoch 167/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.0828
- accuracy: 0.9752 - val_loss: 2.2353 - val_accuracy: 0.5522
Epoch 168/200
3787/3787 [==============================] - 754s 199ms/step - loss: 0.0822
- accuracy: 0.9704 - val_loss: 2.3095 - val_accuracy: 0.5586
```

```
Epoch 169/200
3787/3787 [==============================] - 756s 200ms/step - loss: 0.0963
- accuracy: 0.9704 - val_loss: 2.3240 - val_accuracy: 0.5309
Epoch 170/200
3787/3787 [==============================] - 754s 199ms/step - loss: 0.0896
- accuracy: 0.9699 - val_loss: 2.2833 - val_accuracy: 0.5565
Epoch 171/200
3787/3787 [==============================] - 754s 199ms/step - loss: 0.0816
- accuracy: 0.9736 - val_loss: 2.3418 - val_accuracy: 0.5672
Epoch 172/200
3787/3787 [==============================] - 755s 199ms/step - loss: 0.0753
- accuracy: 0.9749 - val_loss: 2.3648 - val_accuracy: 0.5416
Epoch 173/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.0946
- accuracy: 0.9702 - val_loss: 2.3801 - val_accuracy: 0.5394
Epoch 174/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.0776
- accuracy: 0.9744 - val_loss: 2.4821 - val_accuracy: 0.5181
Epoch 175/200
3787/3787 [==============================] - 757s 200ms/step - loss: 0.0875
- accuracy: 0.9704 - val_loss: 2.4084 - val_accuracy: 0.5501
Epoch 176/200
3787/3787 [==============================] - 758s 200ms/step - loss: 0.0806
- accuracy: 0.9733 - val_loss: 2.4274 - val_accuracy: 0.5373
Epoch 177/200
3787/3787 [==============================] - 751s 198ms/step - loss: 0.0745
- accuracy: 0.9757 - val_loss: 2.2818 - val_accuracy: 0.5480
Epoch 178/200
3787/3787 [==============================] - 755s 199ms/step - loss: 0.0801
- accuracy: 0.9747 - val_loss: 2.5710 - val_accuracy: 0.5416
Epoch 179/200
3787/3787 [==============================] - 754s 199ms/step - loss: 0.0795
- accuracy: 0.9752 - val_loss: 2.4981 - val_accuracy: 0.5330
Epoch 180/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.0769
- accuracy: 0.9762 - val_loss: 2.5984 - val_accuracy: 0.5352
Epoch 181/200
3787/3787 [==============================] - 751s 198ms/step - loss: 0.0793
- accuracy: 0.9731 - val_loss: 2.4714 - val_accuracy: 0.5352
Epoch 182/200
3787/3787 [==============================] - 756s 200ms/step - loss: 0.0733
- accuracy: 0.9765 - val_loss: 2.4161 - val_accuracy: 0.5458
Epoch 183/200
3787/3787 [==============================] - 752s 199ms/step - loss: 0.0788
- accuracy: 0.9736 - val_loss: 2.4608 - val_accuracy: 0.5288
Epoch 184/200
```

```
3787/3787 [==============================] - 754s 199ms/step - loss: 0.0636
- accuracy: 0.9805 - val_loss: 2.3791 - val_accuracy: 0.5565
Epoch 185/200
3787/3787 [==============================] - 755s 199ms/step - loss: 0.0713
- accuracy: 0.9770 - val_loss: 2.4941 - val_accuracy: 0.5309
Epoch 186/200
3787/3787 [==============================] - 753s 199ms/step - loss: 0.0777
- accuracy: 0.9752 - val_loss: 2.4507 - val_accuracy: 0.5288
Epoch 187/200
3787/3787 [==============================] - 751s 198ms/step - loss: 0.0730
- accuracy: 0.9741 - val_loss: 2.5034 - val_accuracy: 0.5416
Epoch 188/200
3787/3787 [==============================] - 758s 200ms/step - loss: 0.0790
- accuracy: 0.9712 - val_loss: 2.5797 - val_accuracy: 0.5181
Epoch 189/200
3787/3787 [==============================] - 757s 200ms/step - loss: 0.0755
- accuracy: 0.9731 - val_loss: 2.5465 - val_accuracy: 0.5224
Epoch 190/200
3787/3787 [==============================] - 757s 200ms/step - loss: 0.0748
- accuracy: 0.9762 - val_loss: 2.5019 - val_accuracy: 0.5267
Epoch 191/200
3787/3787 [==============================] - 775s 205ms/step - loss: 0.0674
- accuracy: 0.9781 - val_loss: 2.4805 - val_accuracy: 0.5352
Epoch 192/200
3787/3787 [==============================] - 714s 189ms/step - loss: 0.0699
- accuracy: 0.9762 - val_loss: 2.5054 - val_accuracy: 0.5437
Epoch 193/200
3787/3787 [==============================] - 779s 206ms/step - loss: 0.0712
- accuracy: 0.9781 - val_loss: 2.5575 - val_accuracy: 0.5416
Epoch 194/200
3787/3787 [==============================] - 784s 207ms/step - loss: 0.0692
- accuracy: 0.9776 - val_loss: 2.5902 - val_accuracy: 0.5394
Epoch 195/200
3787/3787 [==============================] - 778s 205ms/step - loss: 0.0798
- accuracy: 0.9728 - val_loss: 2.6045 - val_accuracy: 0.5203
Epoch 196/200
3787/3787 [==============================] - 789s 208ms/step - loss: 0.0801
- accuracy: 0.9717 - val_loss: 2.4902 - val_accuracy: 0.5309
Epoch 197/200
3787/3787 [==============================] - 779s 206ms/step - loss: 0.0719
- accuracy: 0.9747 - val_loss: 2.4496 - val_accuracy: 0.5330
Epoch 198/200
3787/3787 [==============================] - 783s 207ms/step - loss: 0.0826
- accuracy: 0.9710 - val_loss: 2.5951 - val_accuracy: 0.5288
Epoch 199/200
3787/3787 [==============================] - 781s 206ms/step - loss: 0.0654
- accuracy: 0.9791 - val_loss: 2.5058 - val_accuracy: 0.5480
```

```
Epoch 200/200
3787/3787 [==============================] - 603s 159ms/step - loss: 0.0559
- accuracy: 0.9831 - val_loss: 2.4956 - val_accuracy: 0.5501
Wall time: 1d 18h 19min 19s
```