LAPORAN UAS

ANALISIS BIG DATA

Disusun Oleh:

Salsabilla Octavianingrum	21091397005
Shandy Ilham Alamsyah	21091397015
Affandika Febrian Putra Yunanto	21091397030

PROGRAM STUDI D4 MANAJEMEN INFORMATIKA

FAKULTAS VOKASI

UNIVERSITAS NEGERI SURABAYA

2024

BAB I
PENDAHULUAN

A. Pengertian DeepFace

DeepFace adalah sistem pengenalan wajah berbasis *deep learning* yang dikembangkan oleh Facebook (sekarang Meta) pada tahun 2014. Sistem ini menggunakan *Convolutional Neural Networks* (CNN) untuk mengenali wajah manusia dengan akurasi yang sangat tinggi, bahkan lebih baik daripada kemampuan manusia dalam beberapa pengujian. Dalam uji coba terhadap dataset Labeled Faces in the Wild (LFW), DeepFace mencapai akurasi lebih dari 97% . DeepFace dilatih menggunakan dataset besar yang mencakup jutaan gambar wajah, yang memungkinkan sistem ini untuk mengenali wajah dalam berbagai kondisi pencahayaan, sudut pandang, dan ekspresi. Facebook kemudian menggunakan DeepFace untuk otomatis menandai wajah dalam foto atau video yang diunggah pengguna. Teknologi ini juga diterapkan dalam berbagai bidang, seperti keamanan dan pemantauan, berkat kemampuannya yang efisien dalam memproses gambar wajah secara cepat dan akurat.

1. Fitur Utama DeepFace:

a. Teknologi Deep Learning: DeepFace menggunakan arsitektur CNN yang sangat dalam, yang memungkinkan sistem ini untuk mempelajari dan mengekstrak fitur wajah secara otomatis tanpa memerlukan fitur manual. Ini memungkinkan DeepFace untuk mengenali wajah dalam berbagai kondisi dan dari berbagai sumber gambar.

b. Akurasi Tinggi: Salah satu fitur utama DeepFace adalah tingkat akurasi pengenalan wajahnya yang sangat tinggi. Dalam uji coba pada dataset standar seperti Labeled Faces in the Wild (LFW), DeepFace mencapai akurasi lebih dari 97%, yang pada saat itu lebih baik daripada sistem pengenalan wajah lainnya yang ada.

c. Pelatihan dengan Data Besar: DeepFace dilatih menggunakan dataset yang sangat besar, yang mencakup jutaan gambar wajah dari berbagai individu dengan berbagai variasi pencahayaan, ekspresi, dan sudut pandang. Pelatihan ini memungkinkan DeepFace untuk mengenali wajah manusia dengan sangat baik dalam kondisi dunia nyata.

d. Penerapan dalam Media Sosial: Facebook menggunakan DeepFace untuk memungkinkan pengenalan wajah otomatis dalam foto dan video yang diunggah oleh pengguna. Ini memungkinkan Facebook untuk memberikan rekomendasi tag otomatis pada teman-teman pengguna.

e. Pengenalan Wajah yang Cepat dan Efisien: DeepFace tidak hanya akurat, tetapi juga efisien. Teknologi ini dirancang untuk bekerja dengan cepat, bahkan pada dataset yang besar sekalipun, memungkinkan pengenalan wajah dalam waktu yang sangat singkat.

2. Cara Kerja DeepFace:

a. Preprocessing Gambar: Gambar wajah pertama-tama diproses untuk menyeimbangkan pencahayaan dan menyesuaikan ukuran wajah agar sesuai dengan jaringan yang dilatih.

b. Ekstraksi Fitur: Jaringan CNN digunakan untuk mengekstrak fitur wajah seperti bentuk, struktur tulang, dan ekspresi wajah. CNN ini memiliki beberapa lapisan yang masing-masing belajar mengenali pola yang berbeda dalam gambar.

c. Penyocokan: Fitur yang diekstraksi kemudian dibandingkan dengan database gambar wajah untuk menentukan apakah gambar tersebut sesuai dengan wajah yang ada dalam database.

d. Pengenalan Identitas: Setelah proses penyocokan selesai, sistem akan mengidentifikasi wajah berdasarkan hasil perbandingan tersebut.

3. Penerapan DeepFace:

a. Media Sosial: DeepFace digunakan di platform seperti Facebook untuk otomatis menandai wajah dalam foto atau video yang diunggah oleh pengguna.

b. Keamanan dan Verifikasi: DeepFace dapat digunakan dalam sistem verifikasi biometrik, seperti autentikasi melalui pengenalan wajah pada perangkat atau aplikasi keamanan.

c. Pemantauan dan Pengawasan: DeepFace dapat diterapkan dalam sistem pengawasan untuk mengenali individu dalam video atau rekaman CCTV.

B. Kelebihan DeepFace

1. Akurasi Tinggi: DeepFace memiliki akurasi yang sangat tinggi dalam mengenali wajah. Dalam pengujian terhadap dataset Labeled Faces in the Wild (LFW), DeepFace mencapai tingkat akurasi lebih dari 97%, bahkan lebih baik daripada kemampuan manusia

2. Kemampuan Pengenalan di Berbagai Kondisi: DeepFace dapat mengenali wajah dengan sangat baik dalam berbagai kondisi, seperti pencahayaan yang berbeda, sudut pandang yang bervariasi, dan ekspresi wajah yang berubah. Ini membuatnya sangat efektif dalam dunia nyata, di mana kondisi gambar bisa sangat bervariasi.

3. Cepat dan Efisien: DeepFace dapat memproses gambar dengan cepat dan efisien, memungkinkan pengenalan wajah secara real-time pada sistem pengawasan atau aplikasi lainnya.

4. Penggunaan dalam Berbagai Bidang: Selain digunakan di Facebook untuk pengenalan wajah dalam foto, teknologi ini juga dapat diterapkan di berbagai bidang lain, seperti keamanan (misalnya autentikasi wajah pada perangkat) dan pemantauan (misalnya sistem pengawasan menggunakan CCTV).

5. Meningkatkan Pengalaman Pengguna: Dengan kemampuannya untuk otomatis mengenali wajah dalam foto dan video, DeepFace memberikan kenyamanan dan pengalaman pengguna yang lebih baik dalam media sosial, seperti memberi saran tag otomatis di Facebook.

C. Kekurangan DeepFace

1. Masalah Privasi: Penggunaan DeepFace dalam pengenalan wajah dapat menimbulkan masalah privasi, terutama ketika diterapkan dalam sistem pemantauan publik. Tanpa persetujuan eksplisit, teknologi ini bisa digunakan untuk melacak individu secara massal, yang berpotensi melanggar hak privasi seseorang.

2. Memerlukan Dataset yang Besar dan Komputasi yang Tinggi: DeepFace membutuhkan dataset yang besar untuk pelatihan, yang memerlukan sumber daya komputasi yang signifikan, seperti daya pemrosesan yang tinggi dan penyimpanan data yang besar.

3. Kesulitan dalam Menghadapi Wajah yang Tertutup atau Terhalang: Sistem ini mungkin tidak dapat mengenali wajah dengan baik jika sebagian wajah tertutup atau terhalang, misalnya, jika seseorang mengenakan masker atau kacamata besar.
4. Ketergantungan pada Kualitas Gambar: Kinerja DeepFace dapat terpengaruh oleh kualitas gambar atau video yang diunggah. Jika gambar buram atau terlalu gelap, tingkat akurasi pengenalan bisa menurun.
5. Bias Algoritma: DeepFace, seperti banyak sistem pengenalan wajah lainnya, dapat memiliki bias dalam pengenalan wajah berdasarkan faktor seperti ras atau jenis kelamin, yang dapat memengaruhi akurasi pengenalan untuk individu dari kelompok tertentu.

BAB II
METODE

A. Arsitektur DeepFace

Arsitektur DeepFace mengandalkan *Convolutional Neural Networks* (CNN), yang merupakan jenis jaringan saraf tiruan yang sangat efektif dalam mengolah data gambar. DeepFace didesain dengan berbagai lapisan CNN yang mendalam, yang memungkinkan model untuk belajar dan mengenali fitur wajah secara otomatis dari gambar input. Berikut adalah gambaran umum tentang arsitektur DeepFace:

1. Input Layer:

a. Ukuran Input: Gambar wajah yang akan dianalisis diubah menjadi gambar dengan ukuran 152 x 152 piksel. Setiap gambar diproses untuk memiliki format yang konsisten, dengan pencahayaan dan kontras yang disesuaikan agar dapat diterima oleh jaringan.

b. Preprocessing: DeepFace menerapkan beberapa teknik preprocessing pada gambar, seperti perataan wajah dan normalisasi warna, untuk meningkatkan kualitas pengenalan wajah.

2. Pretrained Model (Face Alignment and Preprocessing):

Sebelum gambar wajah diproses oleh CNN, DeepFace menggunakan teknologi *face alignment*, yaitu teknik yang memastikan wajah dalam gambar diorientasikan dengan cara yang sesuai (misalnya, memposisikan mata pada titik yang sama). Untuk meningkatkan akurasi, DeepFace juga melakukan cropping pada wajah agar bagian yang relevan dari gambar (seperti mata, hidung, dan mulut) lebih menonjol.

3. Convolutional Layers:

DeepFace menggunakan beberapa lapisan *convolutional* untuk mengekstraksi fitur wajah dari gambar. Lapisan ini bekerja dengan mendeteksi pola dan tekstur dalam gambar yang berkaitan dengan elemen wajah seperti mata, hidung, bibir, dan struktur wajah secara keseluruhan. Setiap lapisan konvolusional berfungsi untuk mengidentifikasi fitur wajah pada tingkat yang semakin kompleks. Misalnya, lapisan pertama mungkin mendeteksi tepi atau tekstur, sementara lapisan yang lebih dalam mendeteksi bagian wajah yang lebih spesifik.

4. Pooling Layers:

Setelah setiap lapisan konvolusional, terdapat lapisan *pooling* yang digunakan untuk mengurangi dimensi data sambil mempertahankan fitur penting. Proses ini mengurangi kompleksitas perhitungan dan meningkatkan efisiensi jaringan.

5. Fully Connected Layers:

Setelah proses konvolusi dan pooling, DeepFace menggunakan beberapa lapisan *fully connected* untuk menggabungkan fitur yang telah dipelajari dan menghasilkan representasi wajah yang lebih abstrak. Lapisan ini menghubungkan semua neuron pada satu lapisan ke neuron di lapisan berikutnya, yang memungkinkan jaringan untuk membuat keputusan akhir mengenai identitas wajah.

6. **Output Layer:**

Pada tahap akhir, DeepFace menghasilkan output berupa vektor representasi wajah, yang berisi informasi numerik yang menggambarkan karakteristik unik wajah yang dianalisis. Representasi ini kemudian dibandingkan dengan wajah yang ada dalam database untuk melakukan verifikasi atau identifikasi.

7. **Face Verification:**

Setelah gambar wajah diproses, DeepFace dapat melakukan *face verification* (pemeriksaan kesesuaian wajah), dengan membandingkan vektor representasi wajah yang diperoleh dari gambar input dengan vektor representasi wajah yang tersimpan dalam database. Jika keduanya memiliki jarak yang sangat dekat (misalnya, menggunakan *Euclidean distance*), maka wajah tersebut dianggap cocok.

Arsitektur DeepFace mengintegrasikan teknologi CNN dengan berbagai lapisan konvolusi dan pooling untuk mengekstraksi dan mempelajari fitur wajah secara bertahap. Proses ini kemudian diikuti oleh lapisan fully connected untuk menghasilkan representasi wajah yang dapat digunakan untuk verifikasi atau identifikasi. Teknik preprocessing seperti *face alignment* juga digunakan untuk meningkatkan akurasi. DeepFace berhasil mencapai tingkat akurasi yang sangat tinggi dalam pengenalan wajah dengan arsitektur ini, terutama berkat kedalaman model dan penggunaan dataset besar untuk melatih sistem.

B. Proses pelatihan (training)

Proses pelatihan DeepFace melibatkan beberapa langkah penting untuk mengoptimalkan kemampuan model dalam mengenali wajah manusia dengan akurasi yang sangat tinggi. Proses ini menggunakan teknik *deep learning*, khususnya *Convolutional Neural Networks* (CNN), yang dilatih pada dataset besar. Berikut adalah penjelasan rinci mengenai tahapan-tahapan dalam proses pelatihan DeepFace:

1. **Persiapan Dataset**

a. Dataset Wajah: DeepFace dilatih menggunakan dataset wajah besar yang mencakup jutaan gambar wajah dari berbagai individu, dengan variasi kondisi pencahayaan, ekspresi wajah, dan sudut pandang. Salah satu dataset utama yang digunakan adalah dataset yang mencakup foto-foto wajah dari berbagai sumber, termasuk Facebook dan dataset publik lainnya.

b. Labeling dan Anotasi: Setiap gambar wajah dalam dataset diberi label dengan identitas individu yang relevan, sehingga sistem dapat mempelajari hubungan antara gambar dan identitas tersebut.

2. **Preprocessing Data**

Sebelum data diteruskan ke model, beberapa tahap preprocessing dilakukan untuk menyiapkan gambar wajah agar lebih mudah dikenali oleh jaringan:

a. Face Alignment: Gambar wajah diatur untuk memastikan bahwa fitur wajah utama (seperti mata, hidung, dan mulut) terletak di posisi yang konsisten. Hal ini dilakukan dengan memanfaatkan algoritma untuk mendeteksi titik-titik kunci wajah dan mengoreksi posisi wajah pada gambar.

b. Cropping: Area wajah dipotong untuk menghilangkan latar belakang yang tidak relevan dan memusatkan perhatian pada fitur wajah.

c. Normalisasi Pencahayaan: Proses ini menghilangkan ketidakteraturan pencahayaan pada gambar untuk meningkatkan konsistensi visual dan memudahkan pengenalan wajah meskipun dalam kondisi cahaya yang bervariasi.

3. Arsitektur CNN

Model DeepFace dibangun menggunakan *Convolutional Neural Networks* (CNN), yang terdiri dari beberapa lapisan konvolusi dan pooling. Berikut adalah tahapan pengolahan gambar dalam CNN:

a. Convolutional Layers: Lapisan ini bertanggung jawab untuk mengekstraksi fitur-fitur dasar dari gambar, seperti tepi, tekstur, dan bentuk. Fitur-fitur ini akan semakin kompleks seiring dengan kedalaman lapisan CNN.

b. Pooling Layers: Setiap lapisan konvolusi diikuti dengan lapisan pooling untuk mengurangi dimensi data sambil mempertahankan informasi penting. Ini mengurangi beban komputasi dan meningkatkan efisiensi model.

c. Fully Connected Layers: Setelah proses konvolusi dan pooling, hasilnya dilewatkan melalui beberapa lapisan fully connected. Lapisan-lapisan ini menghubungkan neuron dari lapisan sebelumnya untuk menghasilkan representasi wajah yang lebih abstrak.

4. Fungsi Loss dan Optimisasi

Selama pelatihan, DeepFace menggunakan fungsi *loss* untuk mengukur kesalahan antara hasil prediksi dan label yang benar (identitas wajah yang sesuai). Fungsi loss yang digunakan adalah *triplet loss*, yang berfokus pada peminimalan jarak antara gambar wajah yang sama dan memaksimalkan jarak antara gambar wajah yang berbeda.

a. Triplet Loss: Dalam triplet loss, tiga gambar digunakan pada satu waktu: gambar positif (wajah yang sama dengan gambar referensi), gambar negatif (wajah yang berbeda), dan gambar referensi (wajah yang akan diuji). Model dioptimalkan untuk membuat gambar positif lebih dekat satu sama lain dan gambar negatif lebih jauh dari gambar referensi dalam ruang vektor.

b. Proses Backpropagation: Setelah menghitung kesalahan, model menggunakan *backpropagation* untuk memperbarui bobot pada jaringan. Proses ini dilakukan menggunakan algoritma optimasi, seperti *Stochastic Gradient Descent* (SGD), untuk memperkecil nilai loss dan meningkatkan akurasi model.

5. Pelatihan Model

a. Epochs dan Mini-Batches: Pelatihan dilakukan melalui beberapa iterasi (dikenal sebagai *epochs*), di mana model memproses seluruh dataset. Dataset dibagi menjadi *mini-batches*, yang memungkinkan model belajar dari sebagian data pada setiap iterasi, membuat pelatihan lebih efisien dan menghindari overfitting.

b. Pembagian Data: Data biasanya dibagi menjadi tiga set: data pelatihan (untuk melatih model), data validasi (untuk mengevaluasi kinerja model selama pelatihan), dan data pengujian (untuk menguji model setelah pelatihan selesai).

6. Evaluasi dan Fine-Tuning

a. Validasi dan Akurasi: Selama pelatihan, model diuji pada data validasi untuk memonitor kinerjanya. Jika model menunjukkan akurasi yang baik pada data validasi, pelatihan dapat dilanjutkan.

b. Fine-Tuning: Setelah model dilatih pada dataset besar, sering kali dilakukan *fine-tuning*, di mana model disesuaikan lebih lanjut dengan dataset yang lebih spesifik atau masalah tertentu. Fine-tuning ini melibatkan pembelajaran ulang pada lapisan tertentu dari jaringan untuk memperbaiki prediksi.

7. Hasil Pelatihan

Setelah pelatihan selesai, model DeepFace dapat menghasilkan vektor representasi wajah yang unik untuk setiap individu. Representasi ini dapat digunakan untuk pengenalan wajah, di mana model membandingkan wajah yang diuji dengan wajah yang ada di database untuk memverifikasi atau mengidentifikasi identitas seseorang. Proses pelatihan DeepFace melibatkan pengolahan data wajah melalui berbagai tahap, mulai dari preprocessing gambar hingga penerapan CNN dan optimasi menggunakan *triplet loss*. Proses ini memungkinkan DeepFace untuk mencapai tingkat akurasi yang sangat tinggi dalam pengenalan wajah, yang menjadi salah satu alasan mengapa DeepFace berhasil menutup celah dalam pengenalan wajah yang sebelumnya tidak dapat dicapai oleh teknologi lain.

C. Perbedaan DeepFace dengan Teknologi Pengenalan Wajah Lainnya

DeepFace adalah salah satu sistem pengenalan wajah yang dikembangkan oleh Facebook, yang menggunakan teknik *deep learning* dan *Convolutional Neural Networks* (CNN) untuk mengenali dan memverifikasi wajah manusia. Berbeda dengan teknologi pengenalan wajah lainnya, DeepFace menawarkan beberapa keunggulan yang membedakannya dari sistem pengenalan wajah tradisional. Berikut adalah perbedaan antara DeepFace dan teknologi pengenalan wajah lainnya:

1. **Penggunaan Deep Learning dan CNN**

a. DeepFace: Menggunakan arsitektur CNN yang sangat dalam untuk mempelajari fitur wajah secara otomatis. CNN memungkinkan DeepFace untuk mengenali wajah dengan akurasi tinggi dan bekerja dengan baik dalam berbagai kondisi pencahayaan, ekspresi wajah, dan sudut pandang.

b. Teknologi Lain: Sebelum munculnya DeepFace, banyak sistem pengenalan wajah yang mengandalkan teknik ekstraksi fitur manual, seperti *Eigenfaces* dan *Fisherfaces*, yang lebih bergantung pada analisis statistik dan tidak sekompleks CNN dalam mempelajari representasi wajah.

2. Akurasi Pengenalan

a. DeepFace: Dikenal dengan tingkat akurasi yang sangat tinggi. Dalam pengujian terhadap dataset Labeled Faces in the Wild (LFW), DeepFace mencapai akurasi lebih dari 97%, yang lebih baik dibandingkan banyak teknologi pengenalan wajah lainnya pada waktu itu

b. Teknologi Lain: Teknologi pengenalan wajah tradisional seperti *Eigenfaces* dan *Local Binary Patterns* (LBP) biasanya memiliki tingkat akurasi yang lebih rendah, terutama ketika menghadapi variasi pencahayaan, ekspresi wajah, atau sudut pandang yang ekstrem.

3. Penggunaan Dataset yang Besar

a. DeepFace: DeepFace dilatih menggunakan dataset yang sangat besar, yang mencakup jutaan gambar wajah. Pelatihan pada dataset yang besar memungkinkan model untuk mempelajari berbagai variasi wajah dan meningkatkan kemampuan pengenalan secara signifikan.

b. Teknologi Lain: Sebagian besar teknologi pengenalan wajah tradisional menggunakan dataset yang lebih kecil dan terbatas, yang membuatnya lebih rentan terhadap kesalahan pengenalan, terutama ketika diterapkan pada gambar dengan kondisi yang tidak terduga.

4. Face Alignment dan Preprocessing

a. DeepFace: Menggunakan teknik *face alignment* untuk memastikan bahwa wajah dalam gambar diorientasikan dengan cara yang konsisten, dengan memposisikan titik-titik kunci wajah pada posisi yang seragam (misalnya, mata pada titik yang sama). Hal ini meningkatkan akurasi pengenalan wajah.

b. Teknologi Lain: Beberapa sistem pengenalan wajah tradisional mungkin tidak menggunakan teknik *face alignment* secara otomatis, yang dapat mengurangi akurasi jika wajah dalam gambar tidak sejajar dengan cara yang diinginkan.

5. Kemampuan Menghadapi Variasi Wajah

a. DeepFace: Karena dilatih dengan dataset besar yang mencakup variasi sudut pandang, pencahayaan, dan ekspresi wajah, DeepFace memiliki kemampuan lebih baik dalam mengenali wajah dalam berbagai kondisi yang menantang.

b. Teknologi Lain: Banyak teknologi pengenalan wajah sebelumnya, seperti *Eigenfaces* dan *LBP*, kesulitan dalam mengenali wajah jika gambar memiliki pencahayaan yang buruk, sudut pandang yang ekstrim, atau ekspresi wajah yang berubah.

6. Implementasi di Dunia Nyata

a. DeepFace: Teknologi ini telah diterapkan di berbagai aplikasi nyata, terutama di Facebook, di mana ia digunakan untuk otomatis menandai wajah dalam foto atau video yang diunggah pengguna. DeepFace juga memiliki potensi untuk digunakan dalam sistem keamanan dan pemantauan.

b. Teknologi Lain: Teknologi pengenalan wajah sebelumnya tidak seefisien DeepFace dalam menghadapi gambar yang tidak ideal, dan umumnya lebih terbatas dalam penggunaannya.

7. Kecepatan dan Efisiensi

a. DeepFace: Dengan arsitektur CNN yang sangat efisien dan optimasi yang dilakukan melalui pelatihan pada dataset besar, DeepFace dapat memproses gambar wajah dengan cepat, memungkinkan pengenalan wajah secara real-time.

b. Teknologi Lain: Teknologi pengenalan wajah yang lebih lama sering kali lebih lambat dalam memproses gambar, terutama karena bergantung pada fitur manual yang memerlukan lebih banyak waktu untuk ekstraksi dan analisis.

DeepFace membedakan dirinya dengan teknologi pengenalan wajah lainnya melalui penggunaan *deep learning* yang mendalam, kemampuan untuk mengenali wajah dalam berbagai kondisi yang menantang, dan akurasi yang sangat tinggi. Dengan pendekatan berbasis *Convolutional Neural Networks* dan pelatihan pada dataset besar, DeepFace mampu mengatasi banyak keterbatasan yang dihadapi oleh sistem pengenalan wajah tradisional.

BAB III
HASIL DAN PEMBAHASAN

A. MODEL

- Import library dan modul yang dibutuhkan untuk membangun dan melatih model klasifikasi citra menggunakan TensorFlow dan Keras.
  - **Seaborn** dan **Matplotlib** digunakan untuk visualisasi data,
  - **NumPy** digunakan untuk manipulasi array.
  - **TensorFlow** dan **Keras** digunakan untuk pembuatan dan pelatihan model deep learning, dengan mengimpor modul seperti **Conv2D**, **MaxPooling2D**, dan **Dense** untuk membangun model CNN.
  - **ImageDataGenerator** digunakan untuk augmentasi gambar (menambah variasi data).
  - Model yang dibangun terdiri dari lapisan konvolusional, normalisasi batch, aktivasi ReLU, dan lapisan pooling, diakhiri dengan lapisan pemisahan (flatten) dan fully connected (dense).
  - Optimizer yang digunakan adalah **Adam**, dengan **callbacks** untuk checkpoint model dan early stopping untuk mencegah overfitting.
  - **classification_report** dan **confusion_matrix** digunakan setelah pelatihan untuk evaluasi kinerja model.

```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalization, Activation
from keras.models import Model
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.preprocessing.image import img_to_array
from sklearn.metrics import classification_report, confusion_matrix
```

- inisialisasi direktori untuk dataset pelatihan dan pengujian, yang digunakan dalam proses pelatihan model klasifikasi citra, dan inisialisasi kelas label, dimana setiap gambar dalam dataset pelatihan dan pengujian akan dikategorikan ke dalam salah satu kelas label tersebut.

```
train_dir = r'train'
test_dir = r'test'
```

```
class_labels=['Angry','Disgust', 'Fear', 'Happy','Neutral','Sad','Surprise']
```

- mendefinisikan dua generator data untuk augmentasi gambar pada model pelatihan dan validasi. untuk data pelatihan menerapkan berbagai augmentasi seperti rotasi, pergeseran, zoom, dan flip horizontal, sementara data validation hanya melakukan rescaling gambar, dengan 20% data digunakan untuk validasi dalam kedua generator tersebut.

```
# Define the training data generator with additional augmentations
train_datagen = ImageDataGenerator(
    shear_range=0.3,             # Randomly shear images
    zoom_range=0.3,              # Randomly zoom into images
    rotation_range=30,           # Randomly rotate images
    width_shift_range=0.2,       # Randomly shift images horizontally
    height_shift_range=0.2,      # Randomly shift images vertically
    brightness_range=[0.8, 1.2], # Randomly adjust brightness
    rescale=1./255,              # Rescale pixel values to [0, 1]
    horizontal_flip=True,        # Randomly flip images horizontally
    fill_mode='nearest',         # Fill missing pixels after transformations
    validation_split=0.2         # Set aside 20% of the data for validation
)

# Define the validation data generator (simpler augmentations)
validation_datagen = ImageDataGenerator(
    rescale=1./255,              # Rescale pixel values to [0, 1]
    validation_split=0.2         # Set aside 20% of the data for validation
)
```

- mendefinisikan dua generator data yang memuat gambar dari direktori pelatihan dan validasi, lalu mengubah ukurannya menjadi 152x152 piksel, dan mengkonversinya menjadi format RGB. Kemudian menyusun data dalam batch dan mengklasifikasinya.

```
# Training data generator
train_generator = train_datagen.flow_from_directory(
    directory=train_dir,         # Directory containing the training data
    target_size=(152, 152),      # Resize all images to 152x152 pixels
    batch_size=32,               # Number of images per batch
    color_mode="rgb",            # Convert images to RGB
    class_mode="categorical",    # Multi-class classification
    subset="training",           # Use the training subset
    shuffle=True                 # Shuffle the data
)

# Validation data generator
validation_generator = validation_datagen.flow_from_directory(
    directory=test_dir,          # Directory containing the validation data
    target_size=(152, 152),      # Resize all images to 152x152 pixels
    batch_size=32,               # Number of images per batch
    color_mode="rgb",            # Convert images to RGB
    class_mode="categorical",    # Multi-class classification
    subset="validation",         # Use the validation subset
    shuffle=True                 # Shuffle the data
)
```

Output

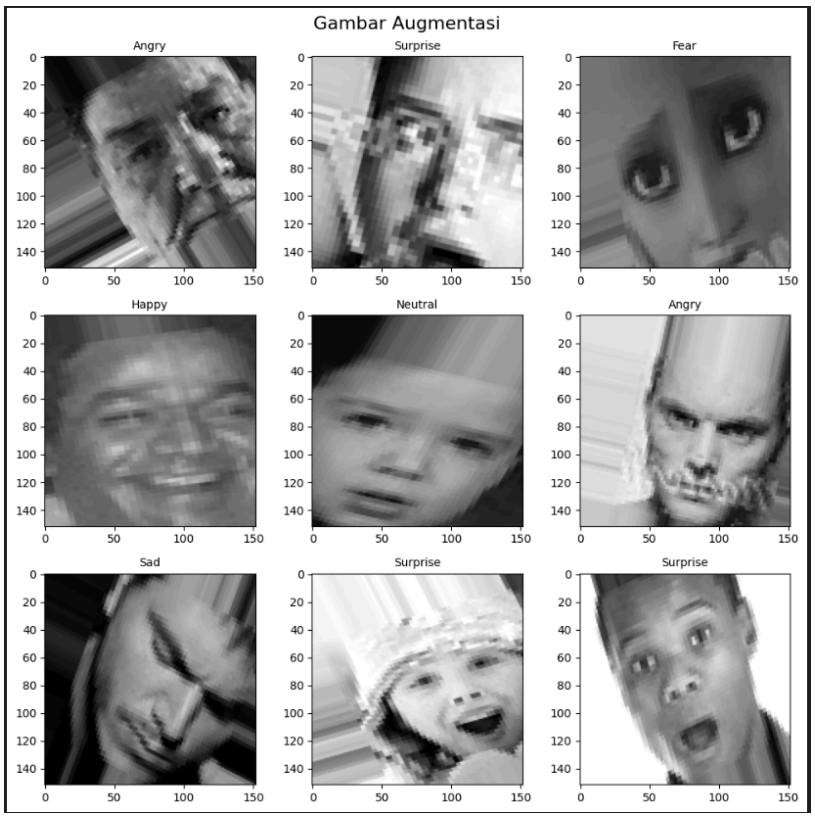
```
...  Found 22968 images belonging to 7 classes.
     Found 1432 images belonging to 7 classes.
```

- Menampilkan beberapa gambar yang sudah di augmentasi sebelumnya.

```python
# Fungsi untuk menampilkan gambar yang diaugmentasi
def display_augmented_images(generator, class_labels, num_images=9):
    """
    Menampilkan gambar augmentasi dengan label kelas.

    Args:
    - generator: Generator data yang mengembalikan gambar dan label.
    - class_labels: List nama kelas sesuai urutan label dari generator.
    - num_images: Jumlah gambar yang ingin ditampilkan.
    """
    # Ambil satu batch gambar dari generator
    data_iter = next(generator)
    images, labels = data_iter  # Gambar dan label

    # Tampilkan beberapa gambar yang diaugmentasi
    plt.figure(figsize=(10, 10))
    plt.suptitle("Gambar Augmentasi", fontsize=16)
    for i in range(num_images):
        plt.subplot(3, 3, i + 1)  # Membuat grid 3x3
        plt.imshow(images[i])
        class_index = labels[i].argmax()  # Cari indeks kelas
        plt.title(class_labels[class_index], fontsize=10)  # Tambahkan nama kelas
        plt.axis('on')  # Nonaktifkan sumbu
    plt.tight_layout()
    plt.show()
```

```python
# Tampilkan gambar yang diaugmentasi
display_augmented_images(train_generator, class_labels, num_images=9)
```

Output



- Konfigurasi model DeepFace, dengan membangun sebuah model yang terinspirasi dari arsitektur DeepFace, lalu menampilkan ringkasan model yang telah dibangun.

```python
# DeepFace model configuration
def build_deepface_model(input_shape=(152, 152, 3)):
    """
    Builds the DeepFace model with the specified input shape.
    :param input_shape: tuple, the shape of the input images (default is (152, 152, 3))
    :return: Keras Model object
    """
    # Define the input layer
    input_layer = Input(shape=input_shape)

    # First convolutional block
    x = Conv2D(32, (11, 11), strides=(4, 4), padding='same')(input_layer)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same')(x)

    # Second convolutional block
    x = Conv2D(16, (9, 9), strides=(1, 1), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same')(x)

    # Third convolutional block
    x = Conv2D(16, (7, 7), strides=(1, 1), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same')(x)

    # Flatten and Fully Connected Layers
    x = Flatten()(x)
    x = Dense(4096, activation='relu')(x)
    x = Dense(4096, activation='relu')(x)
    output_layer = Dense(7, activation='relu')(x)  # Embedding layer

    # Create the model
    model = Model(inputs=input_layer, outputs=output_layer)

    return model

# Example usage
if __name__ == "__main__":
    model = build_deepface_model()
    model.summary()
```

Output

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 152, 152, 3)	0
conv2d (Conv2D)	(None, 38, 38, 32)	11,648
batch_normalization (BatchNormalization)	(None, 38, 38, 32)	128
activation (Activation)	(None, 38, 38, 32)	0
max_pooling2d (MaxPooling2D)	(None, 19, 19, 32)	0
conv2d_1 (Conv2D)	(None, 19, 19, 16)	41,488
batch_normalization_1 (BatchNormalization)	(None, 19, 19, 16)	64
activation_1 (Activation)	(None, 19, 19, 16)	0
max_pooling2d_1 (MaxPooling2D)	(None, 10, 10, 16)	0
conv2d_2 (Conv2D)	(None, 10, 10, 16)	12,560
batch_normalization_2 (BatchNormalization)	(None, 10, 10, 16)	64
activation_2 (Activation)	(None, 10, 10, 16)	0
max_pooling2d_2 (MaxPooling2D)	(None, 5, 5, 16)	0
flatten (Flatten)	(None, 400)	0
dense (Dense)	(None, 4096)	1,642,496
dense_1 (Dense)	(None, 4096)	16,781,312
dense_2 (Dense)	(None, 7)	28,679

Total params: 18,518,439 (70.64 MB)

Trainable params: 18,518,311 (70.64 MB)

Non-trainable params: 128 (512.00 B)

- Konfigurasi model pembelajaran dengan optimizer Adam, fungsi categorical_crossentropy untuk klasifikasi, dan mengevaluasi kinerja model.

```
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

- mendefinisikan dua callback: **ModelCheckpoint** untuk menyimpan model terbaik berdasarkan val_accuracy, dan **EarlyStopping** untuk menghentikan pelatihan jika tidak ada perbaikan pada val_loss selama 10 epoch berturut-turut, guna mencegah overfitting.

```
# Callback untuk menyimpan model terbaik dan early stopping
checkpoint = ModelCheckpoint(
    'fer2013_model.keras',
    monitor='val_accuracy',
    save_best_only=True,
    verbose=1
)

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,
    verbose=1
)
```

- Mulai pelatihan model dengan total 100 epoch, dan menerapkan 2 callback sebelumnya.

```
# Melatih model
history = model.fit(
    train_generator,
    epochs=100,
    validation_data=validation_generator,
    callbacks=[checkpoint, early_stopping]
)
```

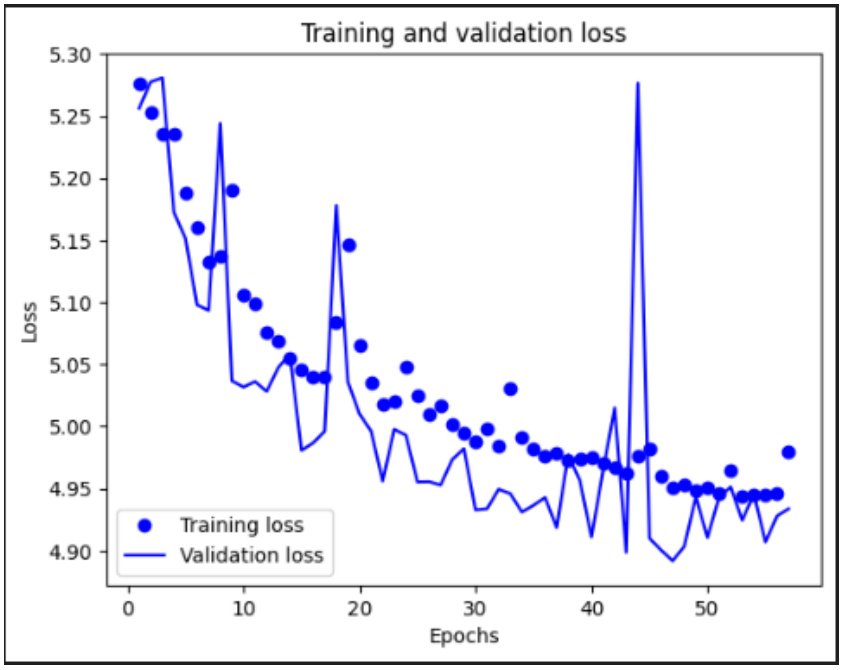
Output

```
c:\Users\SIA\anaconda3\envs\deepface-env\lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWa
  self._warn_if_super_not_called()
Epoch 1/100
718/718 ━━━━━━━━━━━━━━━━━━━━ 0s 821ms/step - accuracy: 0.2453 - loss: 5.2860
Epoch 1: val_accuracy improved from -inf to 0.24860, saving model to fer2013_model.keras
718/718 ━━━━━━━━━━━━━━━━━━━━ 607s 843ms/step - accuracy: 0.2453 - loss: 5.2860 - val_accuracy: 0.2486 - val_loss: 5.2561
Epoch 2/100
718/718 ━━━━━━━━━━━━━━━━━━━━ 0s 462ms/step - accuracy: 0.2595 - loss: 5.2388
Epoch 2: val_accuracy did not improve from 0.24860
718/718 ━━━━━━━━━━━━━━━━━━━━ 335s 466ms/step - accuracy: 0.2595 - loss: 5.2389 - val_accuracy: 0.2444 - val_loss: 5.2775
Epoch 3/100
718/718 ━━━━━━━━━━━━━━━━━━━━ 0s 477ms/step - accuracy: 0.2680 - loss: 5.2668
Epoch 3: val_accuracy improved from 0.24860 to 0.25698, saving model to fer2013_model.keras
718/718 ━━━━━━━━━━━━━━━━━━━━ 348s 485ms/step - accuracy: 0.2680 - loss: 5.2668 - val_accuracy: 0.2570 - val_loss: 5.2808
Epoch 4/100
718/718 ━━━━━━━━━━━━━━━━━━━━ 0s 418ms/step - accuracy: 0.2826 - loss: 5.2056
Epoch 4: val_accuracy improved from 0.25698 to 0.32612, saving model to fer2013_model.keras
718/718 ━━━━━━━━━━━━━━━━━━━━ 304s 424ms/step - accuracy: 0.2826 - loss: 5.2057 - val_accuracy: 0.3261 - val_loss: 5.1725
Epoch 5/100
718/718 ━━━━━━━━━━━━━━━━━━━━ 0s 427ms/step - accuracy: 0.2998 - loss: 5.2792
Epoch 5: val_accuracy improved from 0.32612 to 0.35405, saving model to fer2013_model.keras
718/718 ━━━━━━━━━━━━━━━━━━━━ 311s 434ms/step - accuracy: 0.2998 - loss: 5.2790 - val_accuracy: 0.3541 - val_loss: 5.1514
Epoch 6/100
718/718 ━━━━━━━━━━━━━━━━━━━━ 0s 436ms/step - accuracy: 0.3249 - loss: 5.1153
Epoch 6: val_accuracy improved from 0.35405 to 0.38547, saving model to fer2013_model.keras
718/718 ━━━━━━━━━━━━━━━━━━━━ 316s 440ms/step - accuracy: 0.3249 - loss: 5.1154 - val_accuracy: 0.3855 - val_loss: 5.0978
Epoch 7/100
...
718/718 ━━━━━━━━━━━━━━━━━━━━ 0s 493ms/step - accuracy: 0.4394 - loss: 4.9715
Epoch 57: val_accuracy did not improve from 0.48184
718/718 ━━━━━━━━━━━━━━━━━━━━ 357s 497ms/step - accuracy: 0.4394 - loss: 4.9715 - val_accuracy: 0.4560 - val_loss: 4.9337
Epoch 57: early stopping
```

- Menampilkan plot loss training dan validation

```
# Plot the train and validation loss
train_loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(train_loss) + 1)
plt.plot(epochs, train_loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

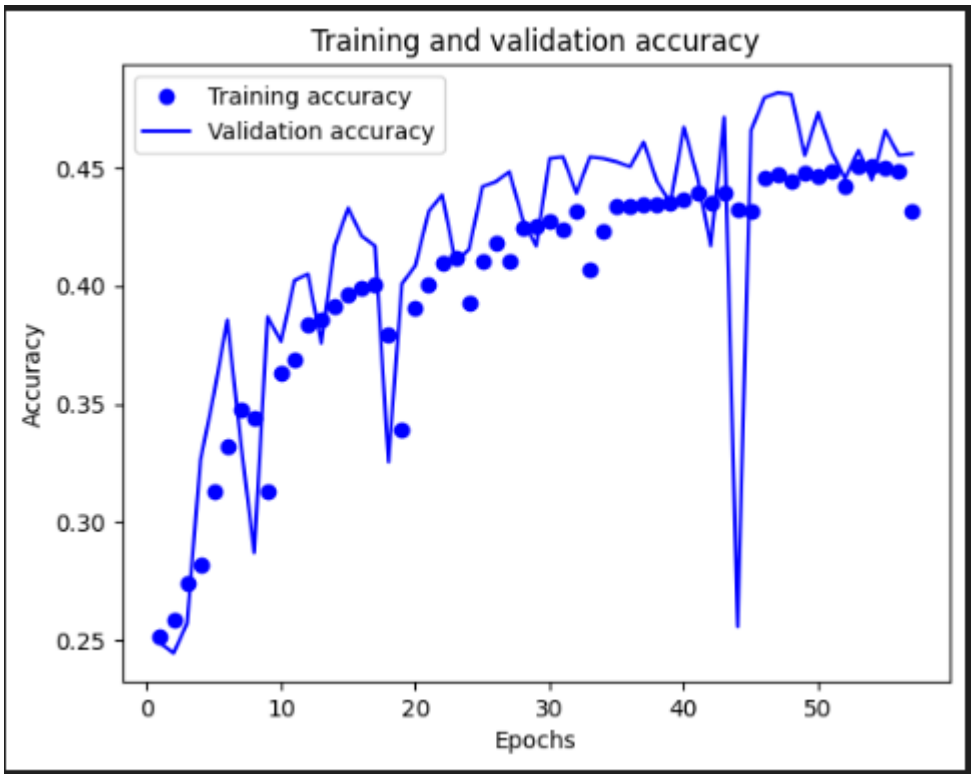
Output



- Menampilkan plot akurasi train dan validation

```
# Plot the train and validation accuracy
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
plt.plot(epochs, train_acc, 'bo', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Output



- Menampilkan classification report

```
# Mengambil label dari generator validasi
val_labels = validation_generator.classes
class_labels = list(validation_generator.class_indices.keys())  # Nama kelas

# Memprediksi data validasi
val_predictions = model.predict(validation_generator, verbose=1)
val_pred_classes = np.argmax(val_predictions, axis=1)

# Classification Report
print("Classification Report:")
print(classification_report(val_labels, val_pred_classes, target_names=class_labels, zero_division=0))
```

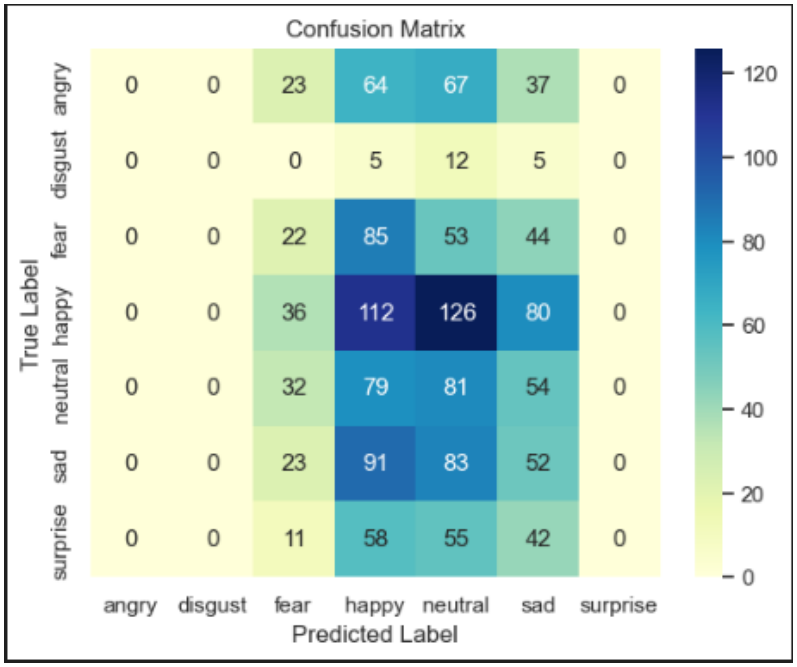
Output

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 4s 70ms/step
Classification Report:
              precision    recall  f1-score   support

       angry       0.00      0.00      0.00       191
     disgust       0.00      0.00      0.00        22
        fear       0.16      0.12      0.14       204
       happy       0.25      0.34      0.29       354
     neutral       0.21      0.41      0.28       246
         sad       0.18      0.22      0.20       249
    surprise       0.00      0.00      0.00       166

    accuracy                           0.21      1432
   macro avg       0.11      0.16      0.13      1432
weighted avg       0.15      0.21      0.17      1432
```

- Menampilkan Confusion Matrix

```
# Get the true labels and predicted labels for the validation set
validation_labels = validation_generator.classes
validation_pred_probs = model.predict(validation_generator)
validation_pred_labels = np.argmax(validation_pred_probs, axis=1)

# Compute the confusion matrix
confusion_mtx = confusion_matrix(validation_labels, validation_pred_labels)
class_names = list(train_generator.class_indices.keys())
sns.set()
sns.heatmap(confusion_mtx, annot=True, fmt='d', cmap='YlGnBu',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

Output



- Menampilkan akurasi dan loss validation

```
# Evaluate the Model
val_loss, val_accuracy = model.evaluate(validation_generator)
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
print(f"Validation Loss: {val_loss * 100:.2f}%")
```

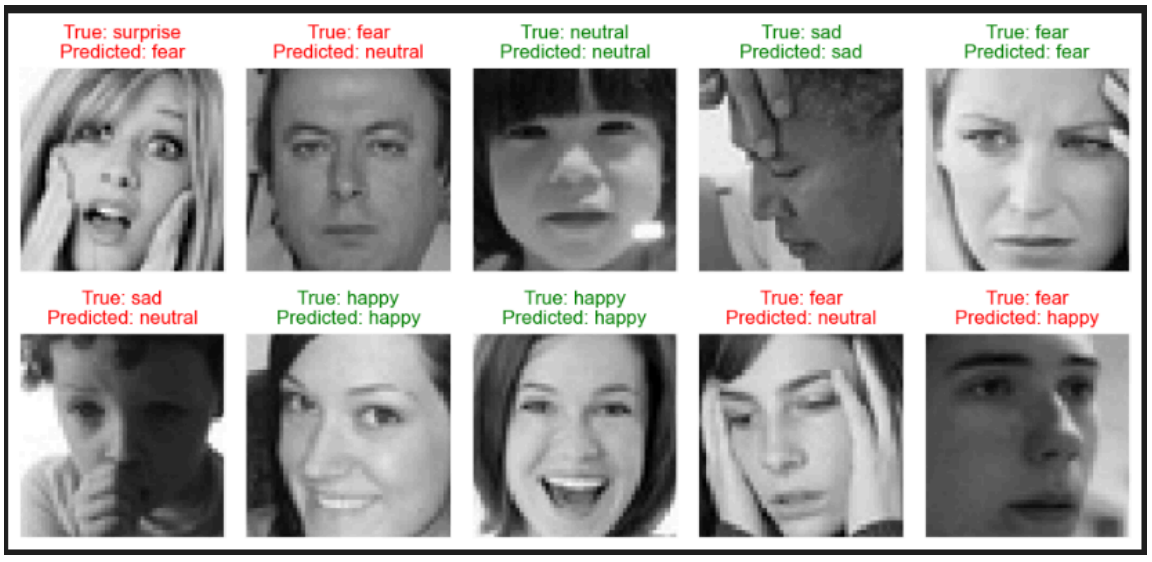
Output

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 3s 72ms/step - accuracy: 0.4467 - loss: 5.1849
Validation Accuracy: 45.60%
Validation Loss: 493.37%
```

- Testing prediksi model, dengan memilih 10 gambar acak dari batch data validasi untuk diprediksi oleh model, dan menampilkan gambar-gambar tersebut bersama dengan label sebenarnya dan prediksi model.

```
# Assuming test_generator and model are already defined
batch_size = validation_generator.batch_size

# Selecting a random batch from the test generator
Random_batch = np.random.randint(0, len(validation_generator) - 1)

# Selecting random image indices from the batch
Random_Img_Index = np.random.randint(0, batch_size, 10)

# Setting up the plot
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(10, 5),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    # Fetching the random image and its label
    Random_Img = validation_generator[Random_batch][0][Random_Img_Index[i]]
    Random_Img_Label = np.argmax(validation_generator[Random_batch][1][Random_Img_Index[i]], axis=0)

    # Making a prediction using the model
    Model_Prediction = np.argmax(model.predict(tf.expand_dims(Random_Img, axis=0), verbose=0), axis=1)[0]

    # Displaying the image
    ax.imshow(Random_Img.squeeze(), cmap='gray')  # Assuming the images are grayscale
    # Setting the title with true and predicted labels, colored based on correctness
    color = "green" if class_labels[Random_Img_Label] == class_labels[Model_Prediction] else "red"
    ax.set_title(f"True: {class_labels[Random_Img_Label]}\nPredicted: {class_labels[Model_Prediction]}", color=color)

plt.tight_layout()
plt.show()
```

Ouput



B. IMPLEMENTASI PADA WEBSITE

- **Import Library**

```python
import streamlit as st
import numpy as np
import tensorflow as tf
import cv2
from PIL import Image, ImageDraw
from mtcnn import MTCNN
from deepface import DeepFace

st.set_page_config(
    page_title= "Face Emotion Recognition Apps",
    page_icon="🎭"
)

st.title("Face Emotion Recognition")
st.write("Welcome to this emotion prediction project, this project created for Analisis Big Data Assignment")
```

- **Real Time Prediction**

```python
# REAL-TIME PREDICTION
# Load pre-trained model
@st.cache_resource
def load_model():
    model = tf.keras.models.load_model("fer2013_model.keras")
    return model

# Define emotion labels
class_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']

# Preprocess the image to match model input
def preprocess_image(image):
    image = cv2.resize(image, (152,152))  # Resize to the target size
    image = image / 255.0  # Normalize pixel values
    if image.shape[-1] == 1:  # If grayscale, convert to RGB
        image = np.repeat(image, 3, axis=-1)
    image = np.expand_dims(image, axis=0)  # Add batch dimension
    return image

# Real-time emotion detection function
def real_time_emotion_detection():
    st.title("Real-Time Emotion Detection")

    # Initialize webcam
    st.write("Nyalakan Webcam anda untuk prediksi secara Real-time")
    run = st.checkbox("Run Webcam")

    if run:
        # Load model
        model = load_model()
        mtcnn = MTCNN()

                # Start video capture
        cap = cv2.VideoCapture(0)

        # Create a placeholder for the video frame
        frame_placeholder = st.empty()

        while run:
            ret, frame = cap.read()
            if not ret:
                st.warning("Failed to capture video. Please check your webcam.")
                break

            # Convert frame to RGB
            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

            # Detect faces using MTCNN
            detections = mtcnn.detect_faces(rgb_frame)

            # Draw bounding boxes and predict emotion
            for detection in detections:
                x, y, width, height = detection['box']
                face = rgb_frame[max(0, y):y + height, max(0, x):x + width]  # Ensure indices are valid
```

```python
                # Preprocess the detected face
                if face.size > 0:
                    processed_face = preprocess_image(face)
                    predictions = model.predict(processed_face)
                    predicted_label = class_labels[np.argmax(predictions)]

                    # Draw bounding box and label
                    cv2.rectangle(frame, (x, y), (x + width, y + height), (0, 255, 0), 2)
                    cv2.putText(frame, predicted_label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

            # Update the frame in the placeholder
            frame_placeholder.image(frame, channels="BGR", use_container_width=True)

        # Release webcam
        cap.release()
        cv2.destroyAllWindows()


if __name__ == "__main__":
    real_time_emotion_detection()
```

- **Input Prediction**

```python
# UPLOAD TO PREDICT
# Load pre-trained model
@st.cache_resource
def load_model():
    model = tf.keras.models.load_model("fer2013_model.keras")  # Replace with your model file path
    return model

# Define emotion labels
class_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']

# Preprocess the image to match model input
def preprocess_image(image):
    image = image.resize((152, 152))  # Resize to the target size
    image = np.array(image) / 255.0  # Normalize pixel values
    if image.shape[-1] == 1:  # If grayscale, convert to RGB
        image = np.repeat(image, 3, axis=-1)
    image = np.expand_dims(image, axis=0)  # Add batch dimension
    return image

# Streamlit app
def main():
    st.title("Upload Image to Predict")

    st.write("Unggah gambar untuk mendeteksi emosi")

    uploaded_file = st.file_uploader("Choose an image file", type=["jpg", "jpeg", "png"])

    if uploaded_file is not None:
        # Display the uploaded image
        image = Image.open(uploaded_file)

        # Initialize MTCNN
        mtcnn = MTCNN()

        # Detect faces
        image_np = np.array(image)
        detections = mtcnn.detect_faces(image_np)

        # Draw bounding boxes and landmarks
        draw = ImageDraw.Draw(image)
        for detection in detections:
            x, y, width, height = detection['box']
            draw.rectangle([x, y, x + width, y + height], outline="red", width=2)

            for key, point in detection['keypoints'].items():
                draw.ellipse([point[0] - 2, point[1] - 2, point[0] + 2, point[1] + 2], fill="red")

        st.image(image, caption="Detected Faces with Landmarks", use_container_width=True)

        # Preprocess the image
        processed_image = preprocess_image(image)

        # Load model and make prediction
        model = load_model()
        predictions = model.predict(processed_image)
```

```python
        # Get the predicted emotion and its probability
        predicted_label_index = np.argmax(predictions)
        predicted_label = class_labels[predicted_label_index]
        confidence = np.max(predictions) * 100  # Convert to percentage

        st.write(f"Predicted Emotion: **{predicted_label}**")
        st.write(f"Confidence: **{confidence:.2f}%**")

if __name__ == "__main__":
    main()
```