APENDIXIES

Appendix 01. Curriculum Vitae

CURRICULUM VITAE

Putu Ananda Adi Savitri was born in Denpasar on September 23rd, 2004. The author was born to I Made Dana and Gusti Ketut Arwati. She is Indonesian and Hindu. She currently resides in Permata Anyar Housing Complex, Permata Biru Alley, C2/26, Badung Regency, Bali Province. She completed her elementary education at Imanuel Elementary School and graduated in 2016. She then continued her education at SMP Negeri 2 Kuta Utara and graduated in 2019. In 2022, she graduated from SMA Negeri 1 Kuta Utara with a major in Mathematics and Natural Sciences and pursued a bachelor's degree in Informatics Engineering at Ganesha University of Education. She is currently studying Computer Science at Ganesha University of Education. During her studies, she developed an interest in artificial intelligence, digital image processing, and data analysis. She is writing a thesis entitled "Comparison of CNN and CNN-LSTM Performance in Facial Expression Classification Based on the FER2013 Dataset" as one of the requirements for her Bachelor of Computer Science degree. She hopes that the results of this research will benefit the development of facial expression recognition technology and serve as a reference for future deep learning-based research.

Appendix 02. CNN Training code

```
#Import Library
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    Conv2D,
    MaxPooling2D,
    Flatten,
    Dense,
    Dropout,
    BatchNormalization
)

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    ConfusionMatrixDisplay
)

#Load Dataset FER2013
data = pd.read_csv('fer2013.csv')
```

```
pixels = data['pixels'].tolist()

X = np.array([
    np.fromstring(pixel, sep=' ').reshape(48, 48)
    for pixel in pixels
])

#Reshape menjadi (48,48,1)
X = np.expand_dims(X, -1)

#Normalisasi
X = X / 255.0

#Label
y = to_categorical(data['emotion'], num_classes=7)

#Split Dataset
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    stratify=y,
    random_state=42
)

print("Jumlah Data Training :", X_train.shape)
print("Jumlah Data Testing  :", X_test.shape)

#Class Weight
y_train_int = np.argmax(y_train, axis=1)

class_weights = compute_class_weight(
```

```
    class_weight='balanced',
    classes=np.unique(y_train_int),
    y=y_train_int
)

class_weights = dict(enumerate(class_weights))

print("Class Weight:")
print(class_weights)

#Data Augmentation
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.05,
    height_shift_range=0.05,
    zoom_range=0.1,
    horizontal_flip=True
)

datagen.fit(X_train)

#Build CNN Model
model = Sequential([

    #Conv Block 1
    Conv2D(
        64,
        (3,3),
        activation='relu',
        input_shape=(48,48,1)
    ),
    BatchNormalization(),
```

```
MaxPooling2D(pool_size=(2,2)),

#Conv Block 2
Conv2D(
    128,
    (3,3),
    activation='relu'
),
BatchNormalization(),
MaxPooling2D(pool_size=(2,2)),

#Conv Block 3
Conv2D(
    256,
    (3,3),
    activation='relu'
),
BatchNormalization(),
MaxPooling2D(pool_size=(2,2)),

#Conv Block 4
Conv2D(
    512,
    (3,3),
    activation='relu'
),
BatchNormalization(),
MaxPooling2D(pool_size=(2,2)),

#Fully Connected Layer
Flatten(),
Dense(
```

```
        128,
        activation='relu'
    ),

    Dropout(0.5),

    #Output Layer
    Dense(
        7,
        activation='softmax'
    )
])

#Compile Model
model.compile(
    optimizer=tf.keras.optimizers.Adam(
        learning_rate=5e-5
    ),

    loss=tf.keras.losses.CategoricalCrossentropy(
        label_smoothing=0.1
    ),

    metrics=['accuracy']
)

#Ringkasan Model
model.summary()

#Callback
#Early Stopping
early_stop = EarlyStopping(
```

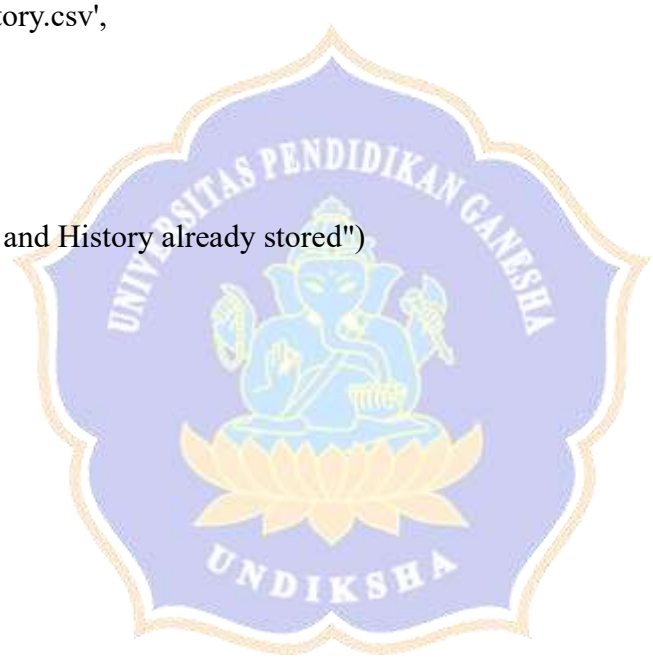
```
    monitor='val_loss',
    patience=12,
    restore_best_weights=True
)

#Reduce Learning Rate
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=5,
    min_lr=1e-6
)

#Training Model
history = model.fit(

    datagen.flow(
        X_train,
        y_train,
        batch_size=64
    ),

    validation_data=(X_test, y_test),

    epochs=100,

    callbacks=[
        early_stop,
        reduce_lr
    ],

    class_weight=class_weights,
```

```
    verbose=1
)

#Evaluasi Model
#Evaluasi testing
test_loss, test_acc = model.evaluate(X_test, y_test)

print(f"\nTesting Accuracy : {test_acc*100:.2f}%")
print(f"Testing Loss     : {test_loss:.4f}")


#Prediksi
y_pred_probs = model.predict(X_test)

y_pred = np.argmax(y_pred_probs, axis=1)
y_true = np.argmax(y_test, axis=1)


#Classification Report
print("\nClassification Report:\n")

print(classification_report(
    y_true,
    y_pred,
    target_names=[
        'Angry',
        'Disgust',
        'Fear',
        'Happy',
        'Sad',
        'Surprise',
```

```
        'Neutral'
    ]
))

#Confusion Matrix
cm = confusion_matrix(y_true, y_pred)

disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=[
        'Angry',
        'Disgust',
        'Fear',
        'Happy',
        'Sad',
        'Surprise',
        'Neutral'
    ]
)

plt.figure(figsize=(8,6))
disp.plot(cmap='Blues', values_format='d')
plt.title("Confusion Matrix")
plt.show()

#Grafik Accuracy & Loss
plt.figure(figsize=(12,5))

#Accuracy
plt.subplot(1,2,1)

plt.plot(
```

```
    history.history['accuracy'],
    label='Training Accuracy'
)

plt.plot(
    history.history['val_accuracy'],
    label='Validation Accuracy'
)

plt.title('Accuracy per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

#Loss
plt.subplot(1,2,2)

plt.plot(
    history.history['loss'],
    label='Training Loss'
)

plt.plot(
    history.history['val_loss'],
    label='Validation Loss'
)

plt.title('Loss per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

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


#Simpan Model & History
#Simpan model
model.save('cnn_fer2013_model.h5')

#Simpan history training
pd.DataFrame(history.history).to_csv(
    'training_history.csv',
    index=False
)

print("\nModel and History already stored")
```



Appendix 03. CNN-LSTM Training code

```
#import library

import numpy as np

import pandas as pd

import os

import tensorflow as tf

import matplotlib.pyplot as plt

import seaborn as sns

from itertools import cycle


#Import Keras

from tensorflow.keras.models import Model, load_model

from tensorflow.keras.layers import Input, LSTM, TimeDistributed, Dense,
Dropout

from tensorflow.keras.utils import to_categorical

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint


#Import untuk Laporan & Plotting

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix, classification_report, roc_curve,
auc


#Import Google Drive

from google.colab import drive
```

```
plt.style.use('seaborn-v0_8-whitegrid')


#Konfigurasi Global

# Konfigurasi Data

IMG_HEIGHT, IMG_WIDTH = 48, 48

NUM_CLASSES = 7

CLASSES = ['Marah', 'Jijik', 'Takut', 'Senang', 'Sedih', 'Terkejut', 'Netral']


#Path

FER2013_CSV_PATH = '/content/drive/My Drive/fer2013.csv'

MODEL_SAVE_PATH = '/content/drive/My
Drive/expression_model_CNNLSTM.h5'


#Fungsi Persiapan Data

def load_and_prepare_data_for_cnnlstm(csv_path):
    """Memuat data FER2013 dan menyiapkannya untuk input CNN-LSTM."""
    try:
        data = pd.read_csv(csv_path)
    except FileNotFoundError:
        print(f"Error: File CSV tidak ditemukan di {csv_path}")
        return None, None, None, None, None, None


    pixels = data['pixels'].tolist()
    faces = []
```

```
    for pixel_sequence in pixels:
        # Reshape menjadi (48 baris, 48 piksel per baris) untuk input sekuensial
        faces.append(np.asarray([int(p) for p in pixel_sequence.split(' ')]).reshape(IMG_HEIGHT, IMG_WIDTH))

    faces = np.asarray(faces)
    faces = faces / 255.0 # Normalisasi

    emotions_int = data['emotion'].astype(int)
    emotions_one_hot = to_categorical(emotions_int, num_classes=NUM_CLASSES)

    #Split data
    X_train_val, X_test, y_train_val, y_test = train_test_split(
        faces, emotions_one_hot, test_size=0.20, random_state=42, stratify=emotions_int)

    y_train_val_int = np.argmax(y_train_val, axis=1)
    X_train, X_val, y_train, y_val = train_test_split(
        X_train_val, y_train_val, test_size=0.125, random_state=42, stratify=y_train_val_int)

    print(f"Bentuk data latih (X_train shape): {X_train.shape}")
    print("Artinya: (jumlah gambar, 48 baris/timesteps, 48 piksel/fitur per baris)")
```

```
    return X_train, y_train, X_val, y_val, X_test, y_test


#Fungsi-Fungsi Model Dan Plotting

def build_cnnlstm_model(input_shape=(IMG_HEIGHT, IMG_WIDTH),
num_classes=NUM_CLASSES):

    """Membangun arsitektur CNN-LSTM."""

    inp = Input(shape=input_shape)


    #TimeDistributed menerapkan layer Dense ke setiap baris (timestep) dari
gambar

    x = TimeDistributed(Dense(64, activation='relu'))(inp)


    # LSTM memproses urutan 48 baris yang sudah diubah menjadi fitur

    x = LSTM(128, return_sequences=False)(x)

    x = Dropout(0.5)(x)


    # Fully Connected Layer untuk klasifikasi akhir

    x = Dense(128, activation='relu')(x)

    x = Dropout(0.5)(x)


    out = Dense(num_classes, activation='softmax')(x)


    model = Model(inputs=inp, outputs=out)

    model.compile(optimizer='adam',
```

```
            loss='categorical_crossentropy',
            metrics=['accuracy'])
    model.summary()
    return model


def plot_training_history(history):
    """Membuat plot untuk training & validation accuracy & loss."""
    acc, val_acc = history.history.get('accuracy', []),
history.history.get('val_accuracy', [])
    loss, val_loss = history.history.get('loss', []), history.history.get('val_loss', [])
    if not acc or not loss: return
    epochs = range(1, len(acc) + 1)
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))
    ax1.plot(epochs, acc, 'bo-', label='Training Accuracy')
    if val_acc: ax1.plot(epochs, val_acc, 'ro-', label='Validation Accuracy')
    ax1.set_title('Training and Validation Accuracy'); ax1.set_xlabel('Epochs');
ax1.set_ylabel('Accuracy'); ax1.legend()
    ax2.plot(epochs, loss, 'bo-', label='Training Loss')
    if val_loss: ax2.plot(epochs, val_loss, 'ro-', label='Validation Loss')
    ax2.set_title('Training and Validation Loss'); ax2.set_xlabel('Epochs');
ax2.set_ylabel('Loss'); ax2.legend()
    plt.suptitle('Grafik Performa Model Selama Pelatihan', fontsize=16); plt.show()


def plot_confusion_matrix(y_true, y_pred_classes, classes):
```

```
    """Membuat plot heatmap untuk confusion matrix."""
    cm = confusion_matrix(y_true, y_pred_classes)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
    plt.title('Confusion Matrix', fontsize=16); plt.ylabel('Label Sebenarnya'); plt.xlabel('Label Prediksi'); plt.show()

def plot_roc_auc_curves(y_true_one_hot, y_pred_probs, n_classes, classes):
    """Membuat plot Kurva ROC untuk setiap kelas (One-vs-Rest)."""
    fpr, tpr, roc_auc = dict(), dict(), dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_true_one_hot[:, i], y_pred_probs[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    plt.figure(figsize=(12, 9))
    colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'green', 'red', 'purple', 'brown'])
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=2,
                 label='Kurva ROC kelas {0} (area = {1:0.2f})'
                 ''.format(classes[i], roc_auc[i]))

    plt.plot([0, 1], [0, 1], 'k--', lw=2, label='No Skill')
```

```
    plt.xlim([0.0, 1.0]); plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate'); plt.ylabel('True Positive Rate')
    plt.title('Kurva ROC Multi-Kelas (One-vs-Rest)', fontsize=16)
    plt.legend(loc="lower right"); plt.show()

if __name__ == '__main__':
    drive.mount('/content/drive')

    print("--- Memuat dan Mempersiapkan Data untuk CNN-LSTM ---")
    X_train, y_train, X_val, y_val, X_test, y_test =
load_and_prepare_data_for_cnnlstm(FER2013_CSV_PATH)

    if X_train is not None:
        print("\n--- Membangun Model CNN-LSTM ---")
        model = build_cnnlstm_model()

        callbacks = [
            EarlyStopping(monitor='val_loss', patience=20,
restore_best_weights=True, verbose=1),
            ModelCheckpoint(MODEL_SAVE_PATH, monitor='val_accuracy',
save_best_only=True, verbose=1)
        ]

        print("\n--- Memulai Pelatihan Model CNN-LSTM ---")
```

```
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=100, # Atur ke angka tinggi, biarkan EarlyStopping bekerja
    batch_size=64,
    callbacks=callbacks
)

print("\n--- Pelatihan Selesai. Mengevaluasi Model pada Data Uji ---")
# Muat kembali model terbaik yang disimpan oleh ModelCheckpoint
best_model = load_model(MODEL_SAVE_PATH)
results = best_model.evaluate(X_test, y_test, verbose=1)
print("Hasil Akhir Uji CNN-LSTM"')
print(f"Loss pada Data Uji: {results[0]:.4f}")
print(f"Akurasi pada Data Uji: {results[1]*100:.2f}%")

#Laporan dan Visualisasi Akhir
print("\nMembuat prediksi untuk laporan dan visualisasi...")
y_pred_probs = best_model.predict(X_test)
y_pred_classes = np.argmax(y_pred_probs, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

print("\Laporan Klasifikasi")
```

```
        print(classification_report(y_true_classes, y_pred_classes,
target_names=CLASSES, digits=4))


        print("\nVisualisasi performa model")

        plot_training_history(history)

        plot_confusion_matrix(y_true_classes, y_pred_classes, classes=CLASSES)

        plot_roc_auc_curves(y_test, y_pred_probs, n_classes=NUM_CLASSES,
classes=CLASSES)

    else:

        print("\nProses dihentikan karena gagal memuat data.")
```

Appendix 04. Live Prediction h5 model

```
#Import Library
import cv2
import numpy as np
import tensorflow as tf
from google.colab import files
from google.colab.patches import cv2_imshow
from IPython.display import display, Javascript
from base64 import b64decode

#upload and load model
files.upload()

model = tf.keras.models.load_model(
    "emotion_cnn_model_optimized.h5"
)

#emotion labels
emotion_labels = [
    "Angry", "Disgust", "Fear",
    "Happy", "Sad", "Surprise", "Neutral"
]

#load face detector
face_cascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
)

#webcam capture
def take_photo():
    js = Javascript('''
        async function takePhoto() {
```

```
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = 'Capture';
      div.appendChild(capture);

      const video = document.createElement('video');
      video.style.border = '2px solid black';
      video.width = 480;
      div.appendChild(video);

      document.body.appendChild(div);

      const stream = await navigator.mediaDevices.getUserMedia({video:
true});
      video.srcObject = stream;
      await video.play();

      await new Promise(resolve => capture.onclick = resolve);

      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      canvas.getContext('2d').drawImage(video, 0, 0);

      stream.getTracks().forEach(track => track.stop());
      div.remove();

      return canvas.toDataURL('image/jpeg');
    }
    takePhoto();
  ''')
  display(js)
```

```
#capture image with button
take_photo()

data = eval_js("data")
img_bytes = b64decode(data.split(',')[1])
img_array = np.frombuffer(img_bytes, np.uint8)
frame = cv2.imdecode(img_array, cv2.IMREAD_COLOR)

#face detection
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(
    gray,
    scaleFactor=1.3,
    minNeighbors=5
)

if len(faces) == 0:
    print("No face detected.")

#preprocess face for model
x, y, w, h = faces[0]

face_gray = gray[y:y+h, x:x+w]
face_gray = cv2.resize(face_gray, (48, 48))
face_gray = face_gray / 255.0
face_gray = face_gray.reshape(1, 48, 48, 1)

#predict emotion
predictions = model.predict(face_gray)[0]
dominant_idx = np.argmax(predictions)
dominant_emotion = emotion_labels[dominant_idx]
```

```
#draw result
result = frame.copy()

cv2.rectangle(result, (x, y), (x+w, y+h), (0, 255, 0), 2)

cv2.putText(
    result,
    f"Dominant: {dominant_emotion}",
    (20, 40),
    cv2.FONT_HERSHEY_SIMPLEX,
    1,
    (0, 255, 0),
    2
)

y_offset = 90
for i, emotion in enumerate(emotion_labels):
    percent = predictions[i] * 100
    text = f"{emotion}: {percent:.2f}%"

    cv2.putText(
        result,
        text,
        (20, y_offset),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.6,
        (255, 255, 255),
        2
    )
    y_offset += 30

cv2_imshow(result)
```