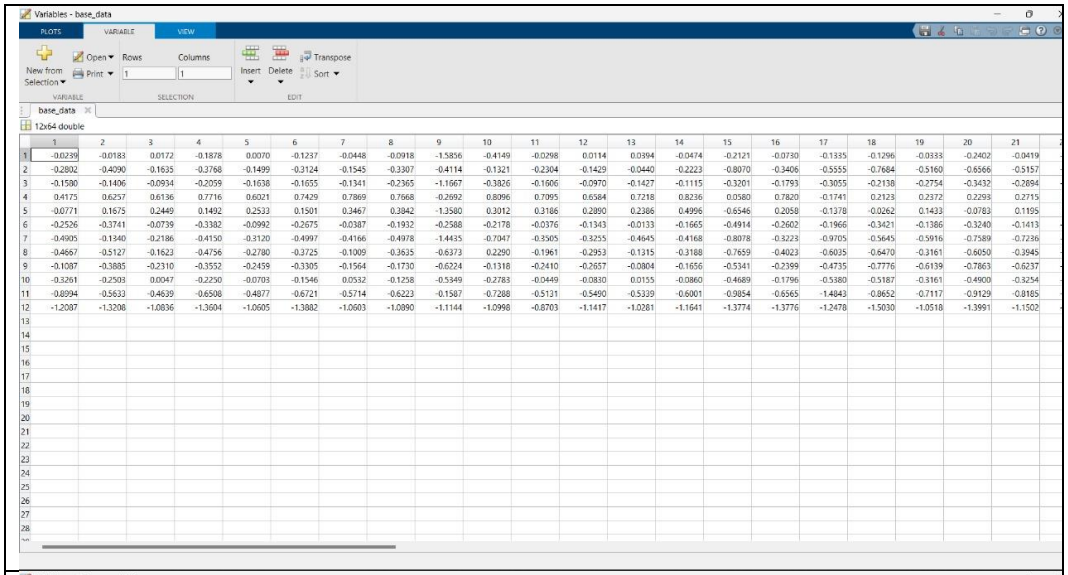
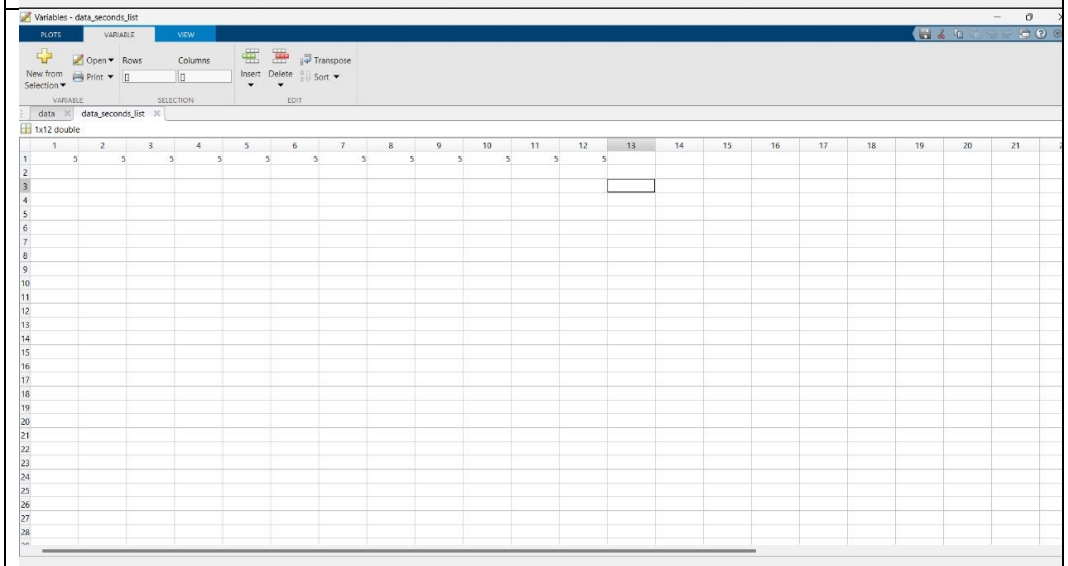
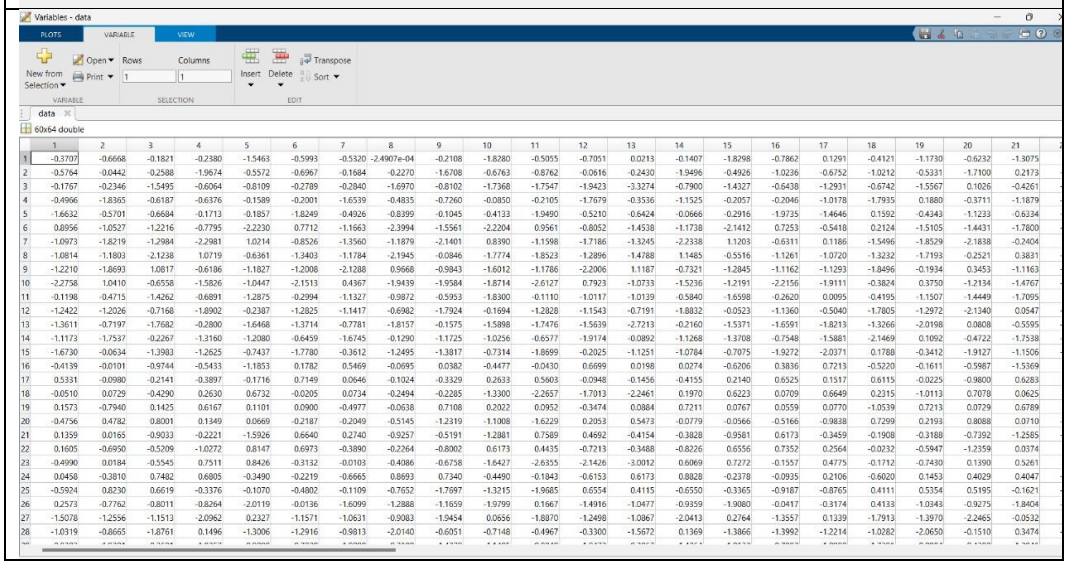
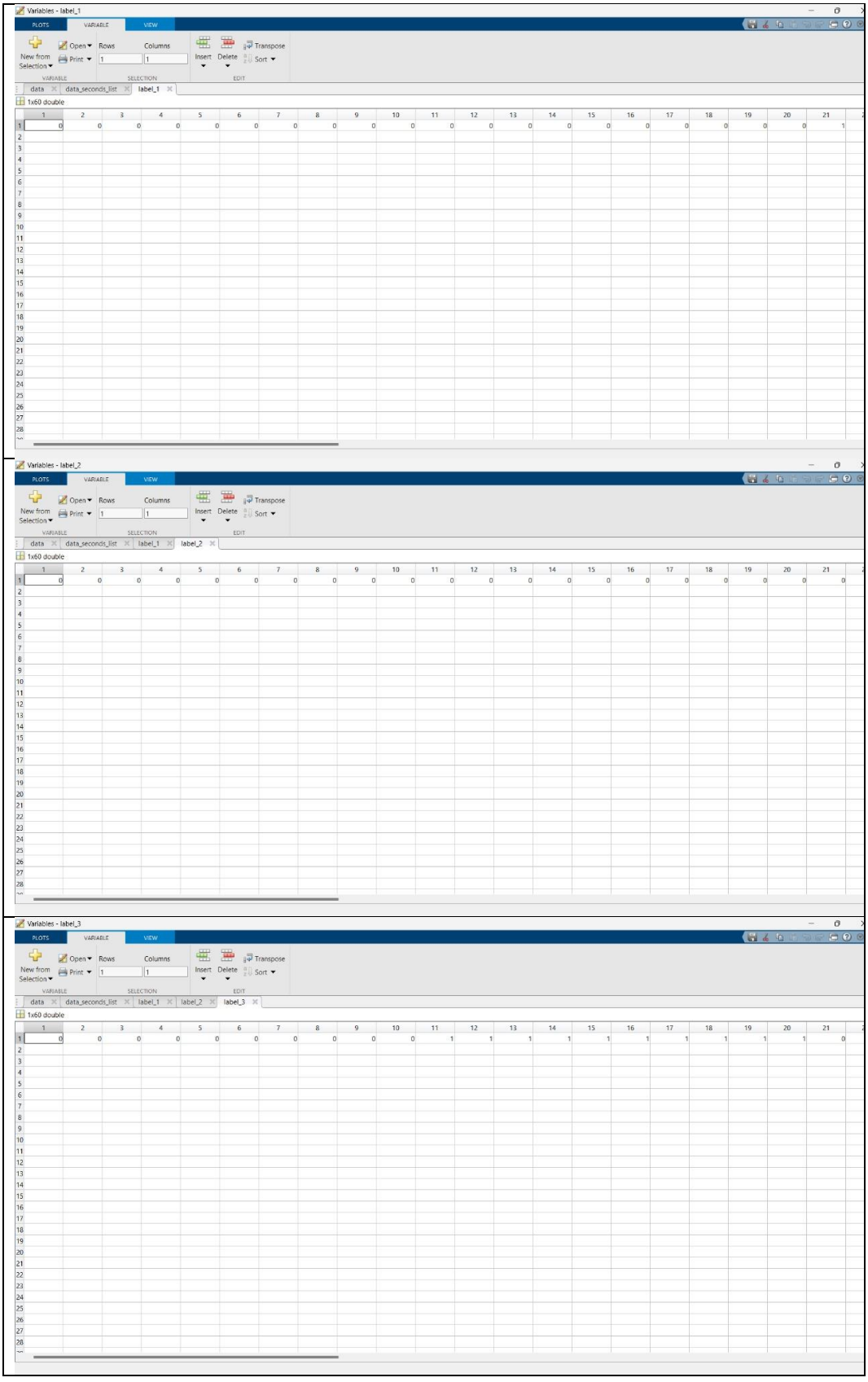
LAMPIRAN

Lampiran 1. Hasil proses data 1D







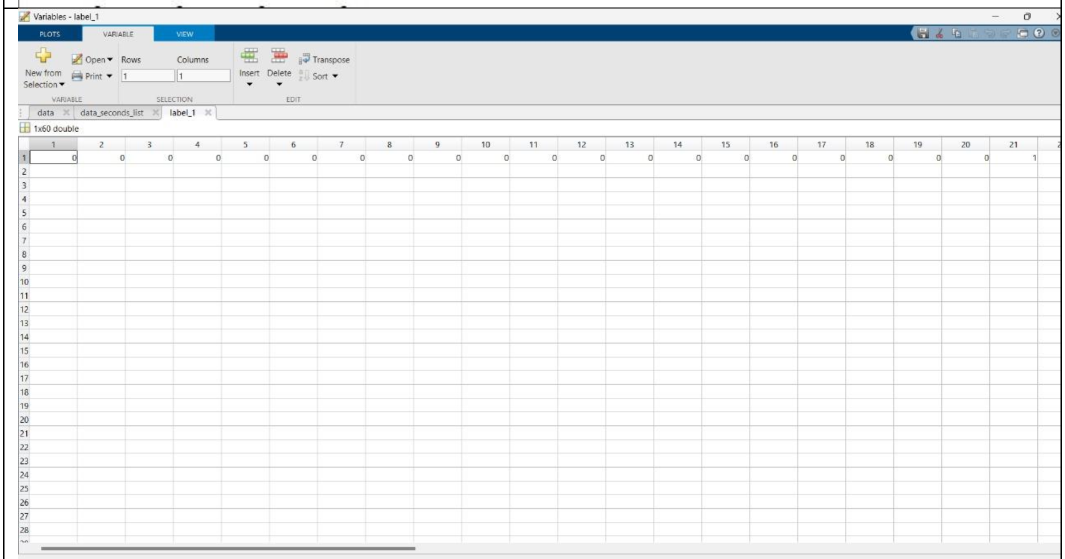

Variables - label_1
PLOTS
VARIABLE
VIEW
New from Selection
Open
Print
Rows
Columns
Insert
Delete
Transpose
Sort
VARIABLE
SELECTION
EDIT
data
data_seconds_list
label_1
1x60 double
Variables - label_2
data
data_seconds_list
label_1
label_2
1x60 double
Variables - label_3
data
data_seconds_list
label_1
label_2
label_3
1x60 double

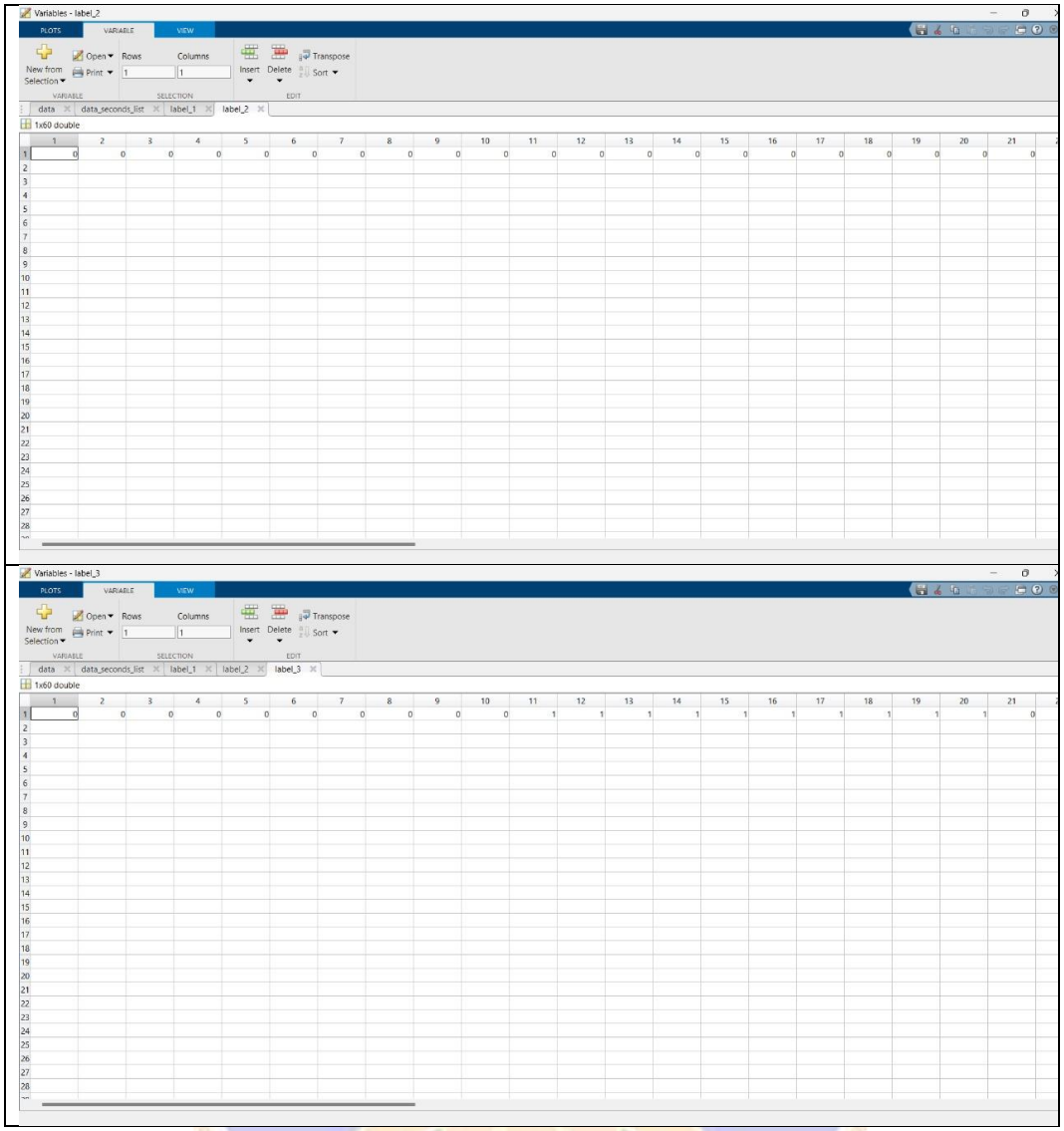
Lampiran 2

```
data
4-D double

val(:,:,1,1) =

     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
```




Variables - label_2
PLOTS
VARIABLE
VIEW
New from Selection
Open
Print
Rows
Columns
Insert
Delete
Transpose
Sort
VARIABLE
SELECTION
EDIT
data
data_seconds_list
label_1
label_2
1x60 double
Variables - label_3
PLOTS
VARIABLE
VIEW
New from Selection
Open
Print
Rows
Columns
Insert
Delete
Transpose
Sort
VARIABLE
SELECTION
EDIT
data
data_seconds_list
label_1
label_2
label_3
1x60 double

Lampiran 3. Proses Segmentasi Data

```
def read_file(file):
    data = sio.loadmat(file)
    return data
trial_signal = data[:, trial][0][:, channel]
base_signal = data[:, trial][0][:300, channel]
```

Lampiran 4. Proses Dekomposisi Data

```python
def butter_bandpass(lowcut, highcut, fs, order=5):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band')
    return b, a

def butter_bandpass_filter(data, lowcut, highcut, fs, order=5):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
    y = lfilter(b, a, data)
    return y

#Dekomposisi Baseline
base_theta = butter_bandpass_filter(base_signal, 4, 8, frequency, order=3)
            base_alpha = butter_bandpass_filter(base_signal, 8,14, frequency, order=3)
            base_beta = butter_bandpass_filter(base_signal,14,31, frequency, order=3)
            base_gamma = butter_bandpass_filter(base_signal,31,45, frequency, order=3)

#Dekomposisi Trial
theta = butter_bandpass_filter(trial_signal, 4, 8, frequency, order=3)
            alpha = butter_bandpass_filter(trial_signal, 8, 14, frequency, order=3)
            beta = butter_bandpass_filter(trial_signal, 14, 31, frequency, order=3)
            gamma = butter_bandpass_filter(trial_signal, 31, 45, frequency, order=3)
```

Lampiran 5. Proses *Feature Extraction* Data

```python
for trial in range(12):
        temp_base_DE = np.empty([0])
        temp_base_theta_DE = np.empty([0])
        temp_base_alpha_DE = np.empty([0])
        temp_base_beta_DE = np.empty([0])
        temp_base_gamma_DE = np.empty([0])

        data_seconds = (data[:, trial][0].shape[0])
        temp_de = np.empty([0,data_seconds])

        for channel in range(16):

#Feature Extraction
base_theta_DE =(compute_DE(base_theta[:100])+compute_DE(base_theta[100:200])+compute_DE(base_theta[200:]))/3
base_alpha_DE =(compute_DE(base_alpha[:100])+compute_DE(base_alpha[100:200])+compute_DE(base_alpha[200:]))/3
base_beta_DE =(compute_DE(base_beta[:100])+compute_DE(base_beta[100:200])+compute_DE(base_beta[200:]))/3
base_gamma_DE =(compute_DE(base_gamma[:100])+compute_DE(base_gamma[100:200])+compute_DE(base_gamma[200:]))/3

temp_base_theta_DE = np.append(temp_base_theta_DE,base_theta_DE)
temp_base_gamma_DE = np.append(temp_base_gamma_DE,base_gamma_DE)
temp_base_beta_DE = np.append(temp_base_beta_DE,base_beta_DE)
temp_base_alpha_DE = np.append(temp_base_alpha_DE,base_alpha_DE)

DE_theta = np.zeros(shape=[0],dtype = float)
DE_alpha = np.zeros(shape=[0],dtype = float)
DE_beta =  np.zeros(shape=[0],dtype = float)
DE_gamma = np.zeros(shape=[0],dtype = float)

for index in range(data_seconds):
                DE_theta =np.append(DE_theta,compute_DE(theta[index*frequency:(index+1)*frequency]))
                DE_alpha =np.append(DE_alpha,compute_DE(alpha[index*frequency:(index+1)*frequency]))
                DE_beta =np.append(DE_beta,compute_DE(beta[index*frequency:(index+1)*frequency]))
                DE_gamma =np.append(DE_gamma,compute_DE(gamma[index*frequency:(index+1)*frequency]))
            temp_de = np.vstack([temp_de,DE_theta])
            temp_de = np.vstack([temp_de,DE_alpha])
            temp_de = np.vstack([temp_de,DE_beta])
            temp_de = np.vstack([temp_de,DE_gamma])
        temp_trial_de = temp_de.reshape(-1,4,)
        decomposed_de = np.vstack([decomposed_de,temp_trial_de])

        temp_base_DE = np.append(temp_base_theta_DE,temp_base_alpha_DE)
        temp_base_DE = np.append(temp_base_DE,temp_base_beta_DE)
        temp_base_DE = np.append(temp_base_DE,temp_base_gamma_DE)
        base_DE = np.vstack([base_DE,temp_base_DE])
        data_seconds_list = np.append(data_seconds_list, data_seconds)
    decomposed_de = decomposed_de.reshape(-1,16,4).transpose([0,2,1]).reshape(-1,4,16).reshape(-1,64)
    print("base_DE shape:",base_DE.shape)
    print("trial_DE shape:",decomposed_de.shape)
    return base_DE, decomposed_de, data_seconds_list
```

Lampiran 6. Proses *Baseline Reduction*

```python
def read_file(file):
    data = sio.loadmat(file)
    return (data['data'], data["base_data"], data["label_1"],
            data["label_2"], data["label_3"], data["data_seconds_list"])

def get_vector_deviation(trial_vec, base_vec):
    return trial_vec / base_vec

def get_dataset_deviation(trial_data, base_data, seconds_list):
    new_dataset = np.empty((0, 64))
    idx = 0
    for i, dur in enumerate(seconds_list[0]):
        for j in range(int(dur)):
            diff = get_vector_deviation(trial_data[idx + j], base_data[i])
            new_dataset = np.vstack([new_dataset, diff.reshape(1, 64)])
        idx += int(dur)
    return new_dataset
```

Lampiran 7. Proses *Feature Representation*

```python
def data_1Dto2D(data, Y=9, X=9):
    data_2D = np.zeros([Y, X])
    data_2D[0] = (0, 0, 0, data[0], 0, data[1], 0, 0, 0)
    data_2D[1] = (0, 0, 0, 0, 0, 0, 0, 0, 0)
    data_2D[2] = (data[8], 0, data[2], 0, 0, 0, data[3], 0, data[9])
    data_2D[3] = (0, 0, 0, 0, 0, 0, 0, 0, 0)
    data_2D[4] = (data[10], 0, data[4], 0, 0, 0, data[5], 0, data[11])
    data_2D[5] = (0, 0, 0, 0, 0, 0, 0, 0, 0)
    data_2D[6] = (data[12], 0, data[6], 0, 0, 0, data[7], 0, data[13])
    data_2D[7] = (0, 0, 0, 0, 0, 0, 0, 0, 0)
    data_2D[8] = (0, 0, 0, data[14], 0, data[15], 0, 0, 0)
    return data_2D

def pre_process(path, use_baseline='yes'):
    sub_len = 16
    data_3D = np.empty((0, 9, 9))

    trial_data, base_data, v_label, a_label, d_label, seconds_list = read_file(path)

    if use_baseline == "yes":
        data = get_dataset_deviation(trial_data, base_data, seconds_list)
    else:
        data = trial_data

    data = preprocessing.scale(data, axis=1)

    for vec in data:
        for band in range(4):
            temp_2D = data_1Dto2D(vec[band*sub_len:(band+1)*sub_len])
            data_3D = np.vstack([data_3D, temp_2D.reshape(1, 9, 9)])

    data_3D = data_3D.reshape(-1, 4, 9, 9)  # (N, 4, 9, 9)
    print("Final shape:", data_3D.shape)
    return data_3D, v_label, a_label, d_label
```

Lampiran 8. Proses *Classification*, dan *Accuracy Calculation*

```python
import os
import numpy as np
import pandas as pd
import scipy.io as sio
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold, train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import precision_score, recall_score, f1_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping

input_height = 9
input_width = 9
time_step = 1
window_size = 1
with_or_not = "with"
inputs = [1, 2, 3, 4]
bands = [i - 1 for i in inputs]
input_channel_num = len(bands) * time_step
dataset_dir = "Dataset_3D/"
mat_files = sorted([f for f in os.listdir(dataset_dir) if f.endswith(".mat") and f.startswith("DE_")])

for file_name in mat_files:
    input_file = file_name.replace(".mat", "")
    print("\n" + "=" * 50)
    print(f"Processing file: {input_file}")
    print("=" * 50)

    data_file = sio.loadmat(os.path.join(dataset_dir, input_file + ".mat"))
    cnn_datasets = data_file["data"]
    label_valence = data_file['label_1'][0]
    label_arousal = data_file['label_2'][0]
    label_dominance = data_file['label_3'][0]

    def preprocess_three_labels(v, a, d):
        mapping = {
            (0, 0, 0): '000',
            (0, 0, 1): '001',
            (0, 1, 0): '010',
            (0, 1, 1): '011',
            (1, 0, 0): '100',
            (1, 1, 0): '110',
        }
        return mapping.get((v, a, d), 'invalid')

    raw_labels = list(map(preprocess_three_labels, label_valence, label_arousal, label_dominance))
    valid_idx = [i for i, lbl in enumerate(raw_labels) if lbl != 'invalid']

    cnn_datasets = cnn_datasets[valid_idx]
    raw_labels = [raw_labels[i] for i in valid_idx]

    encoder = LabelEncoder()
    encoded = encoder.fit_transform(raw_labels)
    labels = to_categorical(encoded)

    cnn_datasets = cnn_datasets.transpose(0, 2, 3, 1)
    cnn_datasets = cnn_datasets[:, :, :, bands]
    cnn_datasets = cnn_datasets.reshape(-1, input_height, input_width, input_channel_num)

    indices = np.arange(len(labels))
    np.random.shuffle(indices)
    cnn_datasets = cnn_datasets[indices]
    labels = labels[indices]

    def build_model(input_shape, num_classes=6, enable_penalty=True):
        model = Sequential()
        model.add(Conv2D(64, kernel_size=4, strides=1, padding='same', activation='relu',
        input_shape=input_shape))
        model.add(Conv2D(128, kernel_size=4, strides=1, padding='same', activation='relu'))
        model.add(Conv2D(256, kernel_size=4, strides=1, padding='same', activation='relu'))
        model.add(Conv2D(64, kernel_size=1, strides=1, padding='same', activation='relu'))
        model.add(Flatten())
        model.add(Dense(1024, activation='relu'))
        model.add(Dropout(0.5))
        if enable_penalty:
            model.add(Dense(num_classes, activation='softmax', activity_regularizer=l2(0.5)))
        else:
            model.add(Dense(num_classes, activation='softmax'))
        return model
```

```python
# ==== K-FOLD TRAINING ====
labels_argmax = np.argmax(labels, axis=1)
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

fold = 1
accuracy_list, precision_list, recall_list, f1_list, epoch_list = [], [], [], [], []

result_dir = os.path.join("CNN Result", f"{with_or_not}_{input_file}_Kernel2x2")
os.makedirs(result_dir, exist_ok=True)

model = build_model(cnn_datasets.shape[1:], labels.shape[1])
model.compile(loss='categorical_crossentropy', optimizer=Adam(1e-4), metrics=['accuracy'])
initial_weights = os.path.join(result_dir, f"initial_model_{input_file}.weights.h5")
model.save_weights(initial_weights)
model.summary()

for train_val_idx, _ in kf.split(cnn_datasets, labels_argmax):
    model.load_weights(initial_weights)

    print(f"\n=== Training Fold {fold} ===")

    X_train, X_val, y_train, y_val = train_test_split(
        cnn_datasets[train_val_idx], labels[train_val_idx],
        test_size=0.1, stratify=labels_argmax[train_val_idx],
        random_state=fold
    )

    early_stop = EarlyStopping(
        monitor='val_accuracy',
        patience=15,
        verbose=1,
        restore_best_weights=True
    )

    hist = model.fit(
        X_train, y_train,
        batch_size=100, epochs=50,
        validation_data=(X_val, y_val),
        callbacks=[early_stop],
        verbose=1
    )

    # === Evaluation ===
    loss, accuracy = model.evaluate(X_val, y_val, verbose=0)
    y_pred = model.predict(X_val)
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_true = np.argmax(y_val, axis=1)

    precision = precision_score(y_true, y_pred_classes, average='macro', zero_division=0)
    recall = recall_score(y_true, y_pred_classes, average='macro', zero_division=0)
    f1 = f1_score(y_true, y_pred_classes, average='macro', zero_division=0)
    total_epochs = len(hist.epoch)

    accuracy_list.append(accuracy)
    precision_list.append(precision)
    recall_list.append(recall)
    f1_list.append(f1)
    epoch_list.append(total_epochs)

    print(f'Fold {fold}: Acc={accuracy:.4f}, Prec={precision:.4f}, Recall={recall:.4f}, F1={f1:.4f}, Epochs={total_epochs}')

    # === Save Model ===

    # === Save History and Metrics to Excel ===
    df_result = pd.DataFrame({
        'epoch': hist.epoch,
        'loss': hist.history['loss'],
        'val_loss': hist.history['val_loss'],
        'accuracy': hist.history['accuracy'],
        'val_accuracy': hist.history['val_accuracy'],
    })

    final_metrics = pd.DataFrame({
        'final_accuracy': [accuracy],
        'final_precision': [precision],
        'final_recall': [recall],
        'final_f1': [f1],
        'total_epoch': [total_epochs]
    })

    with pd.ExcelWriter(os.path.join(result_dir, f'{input_file}_fold_{fold}.xlsx')) as writer:
        df_result.to_excel(writer, sheet_name='training_curve', index=False)
        final_metrics.to_excel(writer, sheet_name='final_metrics', index=False)

    # === Save Plot ===
    plt.figure(figsize=(10, 5))
    plt.plot(hist.history['loss'], label='Train Loss')
    plt.plot(hist.history['val_loss'], label='Val Loss')
    plt.title(f'Loss Curve - Fold {fold}')
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.legend()
    plt.tight_layout()
    plt.savefig(os.path.join(result_dir, f'{input_file}_fold_{fold}_loss.png'))
    plt.close()

    fold += 1

# === Save Summary ===
print("\n=== FINAL AVERAGE METRICS ===")
print(f'Average Accuracy : {np.mean(accuracy_list):.4f}')
print(f'Average Precision: {np.mean(precision_list):.4f}')
print(f'Average Recall   : {np.mean(recall_list):.4f}')
print(f'Average F1-score : {np.mean(f1_list):.4f}')
print(f'Average Epochs   : {np.mean(epoch_list):.2f}')

summary_df = pd.DataFrame({
    'Accuracy': accuracy_list,
    'Precision': precision_list,
    'Recall': recall_list,
    'F1-Score': f1_list,
    'Epoch': epoch_list
})
summary_df.loc['Average'] = summary_df.mean(numeric_only=True)
summary_df.to_excel(os.path.join(result_dir, f'{input_file}_summary.xlsx'))

```

RIWAYAT HIDUP

Made Agastya Maheswara, lahir di Singaraja pada tanggal 16 Desember 2002. Penulis lahir dari pasangan suami istri Bapak Nyoman Suriawan dan Ibu Ketut Sri Indrawati. Penulis merupakan anak kedua dari tiga bersaudara. Penulis berkewarganegaraan Indonesia dan beragama Hindu. Pada tahun 2009 – 2015 penulis menempuh pendidikan Sekolah Dasar di SD Negeri 2 Nyitdah, Kediri, Tabanan. Pada tahun 2015 – 2018 penulis melanjutkan pendidikan Sekolah Menengah Pertama di SMP Negeri 2 Kediri. Selanjutnya, pada tahun 2018 – 2021 penulis menempuh pendidikan Sekolah Menengah Atas di SMA Negeri 1 Tabanan. Pada tahun 2021 penulis melanjutkan pendidikan tinggi di Universitas Pendidikan Ganesha pada Fakultas Teknik dan Kejuruan (FTK), Jurusan Teknik Informatika, Program Studi Sistem Informasi. Dengan tekad, ketekunan, serta semangat belajar yang tinggi, penulis berhasil menyelesaikan skripsi yang berjudul *"IMPLEMENTASI METODE CONTINUOUS CONVOLUTIONAL NEURAL NETWORK UNTUK PENGENALAN POLA PENGANGGE SUARA AKSARA BALI PADA SINYAL ELECTROENCEPHALOGRAM"* pada tahun 2025. Penulis berharap karya ini dapat memberikan kontribusi positif dalam pengembangan penelitian di bidang Sistem Informasi, khususnya pada pengolahan sinyal EEG dan pengenalan pola aksara Bali.