LAMPIRAN

Lampiran 1. Riwayat Hidup

Kartika Nikova lahir di Singaraja pada 23 November 2002. Penulis menetap di kota ini dan memulai pendidikannya di SD Negeri 3 Banjar Jawa serta lulus pada tahun 2014. Penulis kemudian melanjutkan pendidikan ke SMP Negeri 1 Singaraja sampai tahun 2017. Selanjutnya, penulis menyelesaikan jenjang sekolah menengah atas di SMA Negeri 1 Singaraja peminatan Matematika dan Ilmu Pengetahuan Alam (MIPA) pada tahun 2020. Ketertarikan pada bidang teknologi dan data membawa penulis untuk belajar di Program Studi S1 Sistem Informasi, Jurusan Teknik Informatika, Fakultas Teknik dan Kejuruan, Universitas Pendidikan Ganesha. Selama menempuh pendidikan sebagai mahasiswa, penulis aktif terlibat dalam organisasi maupun kegiatan baik di dalam dan luar kampus. Penulis sempat bergabung pada HMJ Teknik Informatika, terpilih sebagai penerima beasiswa Bank Indonesia, mengikuti Program Pertukaran Mahasiswa Merdeka (PMM), serta berbagai kompetisi dan seminar akademik di tingkat nasional dan internasional.

Lampiran 2. Proses Segementasi Data

```
def read_file(file):
      data = sio.loadmat(file)
      return data
#Segmentasi sinyal trial dan sinyal baseline
trial_signal = data[:, trial][0][384:, channel]
base_signal = data[:, trial][0][:384, channel]
```

Lampiran 3. Proses Dekomposisi Data

```
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 4. Proses *Feature Extraction* Data

```
for trial in range(24):
   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]//frequency) - 4
   temp_de = np.empty([0,data_seconds])

   for channel in range(14):
    #FEATURE EXTRACTION
    base_theta_DE
    =(compute_DE(base_theta[:128])+compute_DE(base_theta[128:256
    ])+compute_DE(base_theta[256:]))/3
    base_alpha_DE
    =(compute_DE(base_alpha[:128])+compute_DE(base_alpha[128:256
    ])+compute_DE(base_alpha[256:]))/3
    base_beta_DE
    =(compute_DE(base_beta[:128])+compute_DE(base_beta[128:256])
    +compute_DE(base_beta[256:]))/3
    base_gamma_DE
    =(compute_DE(base_gamma[:128])+compute_DE(base_gamma[128:256
    ])+compute_DE(base_gamma[256:]))/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(-
    1,14,4).transpose([0,2,1]).reshape(-1,4,14).reshape(-1,56)

    print("base_DE shape:",base_DE.shape)
    print("trial_DE shape:",decomposed_de.shape)
    return base_DE, decomposed_de, data_seconds_list
```

Lampiran 5. Proses *Baseline Reduction*

```
def read_file(file):
    file = sio.loadmat(file)
    trial_data = file['data']
    base_data = file["base_data"]
    data_seconds_list = file['data_seconds_list']
    return trial_data,base_data,file["valence_labels"],
file["arousal_labels"], file['dominance_labels'],data_seconds_list

def get_vector_deviation(vector1,vector2):
      return (vector1/vector2)

def get_dataset_deviation(trial_data,base_data,data_seconds_list):
    new_dataset = np.empty([0,56])
    second_now = 0
    for i, seconds in enumerate(data_seconds_list[0]):
       for j in range(int(seconds)):
         new_record = get_vector_deviation(trial_data[j+second_now],
base_data[i]).reshape(1,56)
         new_dataset = np.vstack([new_dataset,new_record])
   second_now += int(seconds)
   return new_dataset

trial_data,base_data,valence_labels,arousal_labels,dominance_labels,
data_seconds_list = read_file(path)
      if y_n=="yes":
       data = get_dataset_deviation (trial_data, base_data,
data_seconds_list)
       data = preprocessing.scale(data,axis=1, with_mean=True,
with_std=True, copy=True)
      else:
       data = preprocessing.scale(trial_data,axis=1, with_mean=True,
with_std=True,copy=True)
```

Lampiran 6. Proses *Feature Representation*

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

#return shape:9*9
return data_2D
```

```
for vector in data:
  for band in range(0,4):
  data_2D_temp = data_1Dto2D (vector[band*sub_vector_len:(band+1)*sub_vector_len])
        data_2D_temp = data_2D_temp.reshape(1,9,9)
        # print("data_2d_temp shape:",data_2D_temp.shape)
        data_3D = np.vstack([data_3D,data_2D_temp])
  data_3D = data_3D.reshape(-1,4,9,9)
  print("final data shape:",data_3D.shape)
  return data_3D,arousal_labels,valence_labels,dominance_labels
```

Lampiran 7. Proses *Classification*, dan *Accuracy Calculation*

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Flatten, Dropout
from tensorflow.keras.regularizers import l2

enable_penalty = True

model = Sequential()
model.add(Conv2D(filters=64, kernel_size=4, strides=1,
padding='same', activation='relu',
input_shape=cnn_datasets.shape[1:]))
model.add(Conv2D(filters=128, kernel_size=4, strides=1,
padding='same', activation='relu'))
model.add(Conv2D(filters=256, kernel_size=4, strides=1,
padding='same', activation='relu'))
model.add(Conv2D(filters=64, kernel_size=1, strides=1,
padding='same', activation='relu'))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))

l2_regularizer = l2(0.5)

if enable_penalty:
    model.add(Dense(6,activation='softmax',
activity_regularizer=l2_regularizer))
else:
    model.add(Dense(6, activation='softmax'))

model.summary()

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import precision_score, recall_score, f1_score
import matplotlib.pyplot as plt

# Define EarlyStopping callback
callback = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)

# Compile the model with required metrics
optimizer = Adam(learning_rate=1e-4)
model.compile(loss='categorical_crossentropy', optimizer=optimizer,
metrics=['accuracy'])
model.save_weights('initial_model.h5')

from sklearn.model_selection import KFold

kf = KFold(n_splits=10)
fold = 1
```

```
for train_idx, test_idx in kf.split(cnn_datasets):
    model.load_weights('initial_model.h5')
    hist = model.fit(cnn_datasets[train_idx], labels[train_idx],
batch_size=128, epochs=50, validation_data=(cnn_datasets[test_idx],
labels[test_idx]), callbacks=[callback])

    # Evaluate the model on the test set
    loss, accuracy = model.evaluate(cnn_datasets[test_idx],
labels[test_idx], verbose=0)
    accuracy_list.append(accuracy)

    # Predict labels for the test set
    y_pred = model.predict(cnn_datasets[test_idx])

    # Convert predicted probabilities to class labels
    y_pred_classes = np.argmax(y_pred, axis=1)

    # Calculate precision, recall, and F1-score
    precision = precision_score(np.argmax(labels[test_idx], axis=1),
y_pred_classes, average='macro')
    recall = recall_score(np.argmax(labels[test_idx], axis=1),
y_pred_classes, average='macro')
    f1 = f1_score(np.argmax(labels[test_idx], axis=1),
y_pred_classes, average='macro')

    # Append metrics to lists
    precision_list.append(precision)
    recall_list.append(recall)
    f1_list.append(f1)

    # Print metrics for the current fold
    print(f'Fold {fold}: Accuracy = {accuracy}, Precision =
{precision}, Recall = {recall}, F1-score = {f1}')

    # Save training history to Excel file
    result_dict = {'epoch': hist.epoch}
    result_dict.update(hist.history)
    result_dict['accuracy'] = accuracy
    result_dict['precision'] = precision
    result_dict['recall'] = recall
    result_dict['f1_score'] = f1
    df_result = pd.DataFrame(result_dict)

    result_dir = "CNN Result\\" + with_or_not + " S1_Kernel3\\"
    if not os.path.isdir(result_dir):
        os.makedirs(result_dir)


df_result.to_excel(f'./{result_dir}{input_file}_fold_{fold}.xlsx',
index=False)
```