4. IMPLEMENTASI SISTEM

Bab ini membahas mengenai implementasi sistem yang dijelaskan pada bab sebelumnya. Implementasi sistem mencakup instalasi, *import library*, dan *dependency* yang digunakan dalam penelitian ini. Proses pertama yang dilakukan adalah melakukan *load dataset training* dan *testing* ke dalam *memory*. Setelah data berhasil di *load* ke dalam *memory*, langkah selanjutnya adalah memulai proses *training* pada model *CNN*. Kemudian melanjutkan proses pengujian pada kumpulan *data testing* untuk mengetahui keakuratan model yang dilatih.

4.1 Instalasi, *Import Library* dan *Dependency*

Dalam sub bab ini membahas mengenai *library* dan *dependency* yang digunakan dalam penelitian ini.

4.1.1 Instalasi dan Konfigurasi *Visual Studio Code*

Langkah pertama yang dilakukan untuk memulai implementasi sistem adalah instalasi *Visual Studio Code*. Berikut langkah-langkah untuk melakukan instalasi *Visual Studio Code*:

1. Buka *link* resmi https://code.visualstudio.com/.
2. *Download installer Visual Studio Code* sesuai dengan sistem operasi yang digunakan.
3. Setelah selesai *download installer*, lakukan proses instalasi dengan mengikuti petunjuk pada *installer Visual Studio Code*.
4. Buka aplikasi *Visual Studio Code* dan *instal python extension* pada bagian *extension*.
5. Buat *folder* baru yang nantinya akan menjadi tempat penyimpanan *file python* yang dibuat.
6. Buat *file* baru dan beri nama *file* dengan format akhiran *.py* sehingga *file* yang tersimpan akan terbaca dalam format *python*.

4.1.2 Instalasi *Library*

Library yang digunakan dalam penelitian ini antara lain adalah *Streamlit* untuk membuat aplikasi *web*, *Tensorflow* serta *Keras* merupakan *library framework machine learning open source* yang digunakan untuk membuat model *deep learning* dan *Scikit-Learn* untuk memvisualisasikan hasil klasifikasi berupa akurasi. Instalasi *library* dilakukan menggunakan perintah "*pip*" yang dijalankan di terminal *Visual Studio Code*.

4.2 *Preprocessing*

Preprocessing yang dilakukan adalah dengan melakukan perubahan seperti *resize* dan *horizontalflipping*.

Segmen Program 4.1 Membuat data *path* dengan label

```
def define_paths(data_dir):
    filepaths = []
    labels = []

    filelist = os.listdir(data_dir)
    for file in filelist:
        label = file.split('_')[-1]
        lable = label.split('.')[0]
        fpath = os.path.join(data_dir, file)
        filepaths.append(fpath)
        labels.append(lable)

    return filepaths, labels
```

Setelah proses mempersiapkan *data path* dan label dalam dua daftar terpisah, kemudian membuat *dataframe*.

Segmen Program 4.2 Membuat *Dataframe*

```
# Menggabungkan data path dengan label ke dalam satu dataframe
def define_df(files, classes):
    Fseries = pd.Series(files, name= 'filepaths')
    Lseries = pd.Series(classes, name='labels')
    return pd.concat([Fseries, Lseries], axis= 1)

def create_df(tr_dir):
    # train and valid dataframe
    files, classes = define_paths(tr_dir)
    df = define_df(files, classes)
    strat = df['labels']
    train_df, valid_df = train_test_split(df, train_size= 0.8, shuffle= True, random_state= 123,
        stratify= strat)

    return train_df, valid_df
```

Setelah membuat *dataframe* dan sebelum data dimasukkan ke dalam model augmentasi data akan diterapkan untuk meningkatkan performa model.

Segmen Program 4.3 *Data Augmentation*

```
def create_gens(train_df, valid_df, batch_size):
    # define model parameters
    img_size = (224, 224)
    channels = 3 # either BGR or Grayscale
    color = 'rgb'
    img_shape = (img_size[0], img_size[1], channels)

    def scalar(img):
        return img

    tr_gen = ImageDataGenerator(preprocessing_function= scalar, horizontal_flip= True)
    ts_gen = ImageDataGenerator(preprocessing_function= scalar)

    train_gen = tr_gen.flow_from_dataframe( train_df, x_col= 'filepaths', y_col= 'labels',
        target_size= img_size, class_mode= 'categorical', color_mode= color, shuffle= True,
        batch_size= batch_size)

    valid_gen = ts_gen.flow_from_dataframe( valid_df, x_col= 'filepaths', y_col= 'labels',
        target_size= img_size, class_mode= 'categorical', color_mode= color, shuffle= True,
        batch_size= batch_size)

    return train_gen, valid_gen
```

4.3 Visualisasi Data

Pada sub bab ini dibahas mengenai menampilkan sampel gambar dari generator data untuk memverifikasi bahwa proses *preprocessing* berjalan dengan benar.

Segmen Program 4.4 Visualisasi Data *Image*

```
def show_images(gen):
    # return classes , images to be displayed
    g_dict = gen.class_indices      # defines dictionary
    classes = list(g_dict.keys())   # defines list of dictionary's (class : string)
    images, labels = next(gen)      # get a batch size samples from the generator

    # calculate number of displayed samples
    length = len(labels)      # length of batch size
    sample = min(length, 25)    # check if sample less than 25 images

    plt.figure(figsize= (20, 20))

    for i in range(sample):
        plt.subplot(5, 5, i + 1)
        image = images[i] / 255      # scales data to range (0 - 255)
        plt.imshow(image)
```

```
        index = np.argmax(labels[i]) # get image index
        class_name = classes[index]  # get class of image
        plt.title(class_name, color= 'blue', fontsize= 12)
        plt.axis('off')

    plt.show()
```

Segmen Program 4.5 Visualisasi Data *Labels*

```
def plot_label_count(df, plot_title):
    # Define needed variables
    vcounts = df['labels'].value_counts()
    labels = vcounts.keys().tolist()
    values = vcounts.tolist()
    lcount = len(labels)

    if lcount > 55:
        print('The number of labels is > 55, no plot will be produced')
    else:
        plot_labels(lcount, labels, values, plot_title)
```

4.4 Implementasi Model *CNN*

Pada sub bab ini dibahas mengenai proses implementasi model *CNN*. Membangun model jaringan saraf dengan menambahkan model dasar yang telah dilatih sebelumnya, lapisan normalisasi *batch*, lapisan *dense* dengan regularisasi dan aktivasi *ReLU*, lapsan *dropout*, dan akhirnya, lapisan *output* dengan aktivasi *softmax* untuk klasifikasi.

Segmen Program 4.6 Model *CNN*

```
model = Sequential([
    base_model, # Pre-trained base model (e.g., a pre-trained ResNet, VGG, etc.)
    BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001),
    Dense(256,
        kernel_regularizer=regularizers.l2(l=0.016), # L2 regularization on weights
        activity_regularizer=regularizers.l1(0.006), # L1 regularization on activations
        bias_regularizer=regularizers.l1(0.006), # L1 regularization on biases
        activation='relu'), # Dense layer with ReLU activation
    Dropout(rate=0.45, seed=123), # Dropout layer with 45% dropout rate
    Dense(class_count, activation='softmax') # Output layer with softmax activation for
classification
])
```

Setelah model *CNN* telah didefinisikan, langkah selanjutnya adalah mengkompilasi model *CNN*. Mengatur parameter-parameter untuk pelatihan model.

Segmen Program 4.7 Kompilasi Model *CNN*

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

4.5 Implementasi Proses *Training*

Pada sub bab ini dijelaskan mengenai proses *training* menggunakan model *CNN* yang sudah di *load* pada sub bab sebelumnya, Fungsi *training* digunakan untuk melakukan *training* pada model sebanyak 25 epoch.

Segmen Program 4.8 Menentukan Proses *Training*

```
batch_size = 8   # Set batch size for training
epochs = 25   # Number of all epochs in training
patience = 3   # Number of epochs to wait to adjust lr if monitored value does not improve
stop_patience = 10   # Number of epochs to wait before stopping training if monitored value does not improve
threshold = 0.9   # If train accuracy is < threshold adjust monitor accuracy, else monitor validation loss
factor = 0.5   # Factor to reduce lr by
ask_epoch = 5   # Number of epochs to run before asking if you want to halt training
batches = int(np.ceil(len(train_gen.labels) / batch_size))   # Number of training batches to run per epoch

callbacks = [
   MyCallback(model=model, patience=patience, stop_patience=stop_patience,
threshold=threshold, factor=factor, batches=batches, epochs=epochs,
ask_epoch=ask_epoch)]
```

Segmen Program 4.9 Proses *Training* Model

```
history = model.fit(
   x=train_gen, epochs=epochs, verbose=0, callbacks=callbacks,
validation_data=valid_gen, validation_steps=None, shuffle=False)
```

4.6 Implementasi Proses *Testing*

Pada sub bab ini dijelaskan mengenai proses *testing* atau prediksi terhadap data baru. Proses ini diawali dengan melakukan prediksi pada *dataset testing* dengan menggunakan model yang sudah di *training*. Hasil *output* dari pengujian berupa *list* hasil identifikasi kelas *diabetes* retina dari model yang sudah melalui proses *training*.

Segmen Program 4.10 Hasil Prediksi Model

```
test_gen = tf.keras.utils.image_dataset_from_directory(
   test_dir,
   labels=None,
   label_mode=None,
   color_mode='rgb',
   image_size=(224, 224),
   shuffle=False
)

pred = np.array(preds)
print(pred)

files = sorted(os.listdir(test_dir))
print(files[:5])

pred_list = []
for i in pred:
   pred_list.append(np.argmax(i))

files_list = []
for f in files:
   files_list.append(f.replace('.png', ''))

result_dict = {'Id': files, 'Category': pred_list}
result = pd.DataFrame(result_dict)
print(result)
```

Setelah model selesai melakukan prediksi terhadap *testing dataset*, langkah selanjutnya adalah untuk menghitung akurasi. *Output* dari program nantinya berupa akurasi dari prediksi yang telah dilakukan.

Segmen Program 4.11 Hasil Akurasi Model

```
_accuracy  = round(accuracy_score(y_true, y_pred) * 100, 2)
_precission = round(precision_score(y_true, y_pred, average='weighted') * 100, 2)
_recall    = round(recall_score(y_true, y_pred, average='weighted') * 100, 2)
_fscore    = round(f1_score(y_true, y_pred, average='weighted') * 100, 2)
```

4.7 Implementasi Program

Pada sub bab ini akan dibahas mengenai program yang akan dijalankan menggunakan *website* dengan *framework streamlit*.

Segmen Program 4.12 Implementasi Program

```
import streamlit as st
import tensorflow as tf
import numpy as np
import cv2
from PIL import Image

# Path model
model_path = 'C:/Users/ASUS/Desktop/Final/Model/Final_Model.h5'

# Memuat model yang telah dilatih sebelumnya
model = tf.keras.models.load_model(model_path)

# Fungsi untuk memuat dan memproses gambar menggunakan OpenCV
def load_and_preprocess_image(img):
    # Mengubah gambar PIL menjadi array numpy
    img_array = np.array(img)

    # Mengubah RGB ke BGR (format warna default OpenCV)
    img_array = cv2.cvtColor(img_array, cv2.COLOR_RGB2BGR)

    # Mengubah ukuran gambar
    img_array = cv2.resize(img_array, (224, 224))  # Sesuaikan berdasarkan ukuran input
model Anda

    # Menambahkan dimensi ekstra (ukuran batch)
    img_array = np.expand_dims(img_array, axis=0)
    return img_array

# Aplikasi Streamlit
st.title("Identifikasi Penyakit Retinopati Diabetik")
st.write("Unggah Gambar Retina dan Klasifikasikan menggunakan Model CNN yang Telah
Dilatih.")

# Pengunggah file
uploaded_file = st.file_uploader("Pilih gambar...", type=["jpg", "jpeg", "png"])

if uploaded_file is not None:
    # Menampilkan gambar yang diunggah
    img = Image.open(uploaded_file)
    st.image(img, caption='Gambar yang Diunggah', use_column_width=True)
```

```
# Memproses gambar menggunakan OpenCV
img_array = load_and_preprocess_image(img)

# Melakukan prediksi
predictions = model.predict(img_array)
predicted_class = np.argmax(predictions, axis=-1)  # Untuk klasifikasi multi-kelas

# Menampilkan hasil prediksi
class_label = [
    'Retina Normal (0), tidak menunjukkan tanda-tanda diabetes',
    'Retinopati Non-Proliferatif (1), menunjukkan tanda-tanda diabetes yang parah',
    'Retinopati Proliferatif (2), menunjukkan tanda-tanda diabetes'
]
hasil_kelas = class_label[predicted_class[0]]
st.write(f'Prediksi kelas: {hasil_kelas}')
```