BAB IV

HASIL DAN PEMBAHASAN

4.1 *Implementasi*

Seluruh langkah perancangan yang tergambar dalam *flowchart* seperti yang terlihat pada Gambar 3.1 *diimplementasikan* dalam kode program menggunakan bahasa pemrograman *Python* versi 3.7 dan dieksekusi dengan bantuan perangkat lunak *Jupyter Notebook*. Detail langkah-langkah dan kode program untuk setiap proses dalam sistem akan dijelaskan pada bagian hasil dan pembahasan ini. Tahap awal melibatkan impor paket-paket yang akan digunakan. Berikut adalah kode untuk impor paket-paket tersebut.

```
Import numpy as np
Import pandas as pd
Import cv2
Import seaborn as sns
Import matlolib.pylot as plt
From sklearn import metrics
From sklearn. Metrics import confusion_matrix
From keras.utils import np_utils
From keras.layers import
Dense.Input,Dropout.GlobalAveragePooling2D,Flatten,Con
v2D,BatchNormalization,Activation,MaxPooling2D
From keras.models import Model, Sequental
From keras.Optimizers import Adam,RMSprop
```

Paket *numpy* digunakan untuk melaksanakan operasi komputasi pada *tipe data numerik* dan dapat diterapkan pada struktur data seperti *vektor* dan *matriks*. Kemudian, paket pandas menyediakan struktur data dan analisis data yang mudah digunakan. Struktur data dasar dalam *Pandas* disebut *DataFrame,* yang

mempermudah dalam membaca file dengan berbagai format. *Pandas* juga berperan dalam pembuatan tabel, transformasi dimensi data, dan pemeriksaan data.

Paket cv2 digunakan untuk memproses gambar, mencakup fungsi membaca, menampilkan, dan menyimpan gambar. Sementara itu, paket *matplotlib* merupakan perpustakaan visualisasi data multiplatform yang dibangun di atas *array NumPy*. Paket ini dapat digunakan untuk memvisualisasikan data dalam format 2D maupun 3D.

Paket *sklearn* digunakan untuk menerapkan pembelajaran mesin standar dan menangani tugas-tugas penambangan data seperti *reduksi dimensi, klasifikasi, regresi,* pengelompokan data, dan pemilihan model. Terakhir, paket keras digunakan untuk mengembangkan model *deep learning,* karena metode yang digunakan dalam penelitian ini adalah *Convolutional Neural Network* (CNN). Keras dapat dijalankan di atas f*ramework* kecerdasan buatan seperti *TensorFlow, Microsoft Cognitive Toolkit,* dan *Theano.* Tabel 4.1 berisi nama-nama paket dan fungsinya dalam penelitian ini.

Tabel 4.1 packages dan kegunaanya.

No	Package	Kegunaan
1	NumPy	Pemrosesan matriks karena data citra akan menjadi sebuah matriks
2.	Pandas	Membuat data frame yang terdiri dari data informasi citra dan data label citra
3.	OpenCV	Pembacaan data citra
4.	Matplotlib	Membuat plot dari hasil penelitian
5.	Keras	Untuk menggunakan CNN

4.2 Mengubah ukuran data

Dataset KDEF memiliki dimensi 256 x 256 piksel, yang tidak dapat diolah oleh model penelitian ini karena model menerima masukan berukuran 48x48 piksel. Oleh karena itu, perlu dilakukan penyesuaian ukuran pada dataset KDEF. Gambar 4.2 digunakan untuk mengubah dimensi gambar dari 256x256 piksel menjadi 48x48 piksel tanpa melakukan pemotongan pada gambar asli data tersebut.

```
path0 = 'E:/TA KU/Dataset/kdef/0'
path1 = 'E:/TA KU/Dataset/kdef/1'
path2 = 'E:/TA KU/Dataset/kdef/2'
path3 = 'E:/TA KU/Dataset/kdef/3'
path4 = 'E:/TA KU/Dataset/kdef/4'
path5 = 'E:/TA KU/Dataset/kdef/5'
path6 = 'E:/TA KU/Dataset/kdef/6'
images0 = [os.path.join(path0,f) for f in os.listdir(path0) if
os.path.isfile(os.path.join(path0,f))]
images1 = [os.path.join(path1,f) for f in os.listdir(path1) if
os.path.isfile(os.path.join(path1,f))]
23
images2 = [os.path.join(path2,f) for f in os.listdir(path2) if
os.path.isfile(os.path.join(path2,f))]
images3 = [os.path.join(path3,f) for f in os.listdir(path3) if
os.path.isfile(os.path.join(path3,f))]
images4 = [os.path.join(path4,f) for f in os.listdir(path4) if
os.path.isfile(os.path.join(path4,f))]
images5 = [os.path.join(path5,f) for f in os.listdir(path5) if
os.path.isfile(os.path.join(path5,f))]
images6 = [os.path.join(path6,f) for f in os.listdir(path6) if
os.path.isfile(os.path.join(path6,f))]
width,height = 48,48
i = 0
while i < 70:
pth = images0[i]
img = cv2.imread(pth)
imgresize = cv2.resize(img, (width, height))
new_path = '%s/kdef00%s.png' % (path0, i)
io.imsave(new_path, imgresize)
i += 1
```

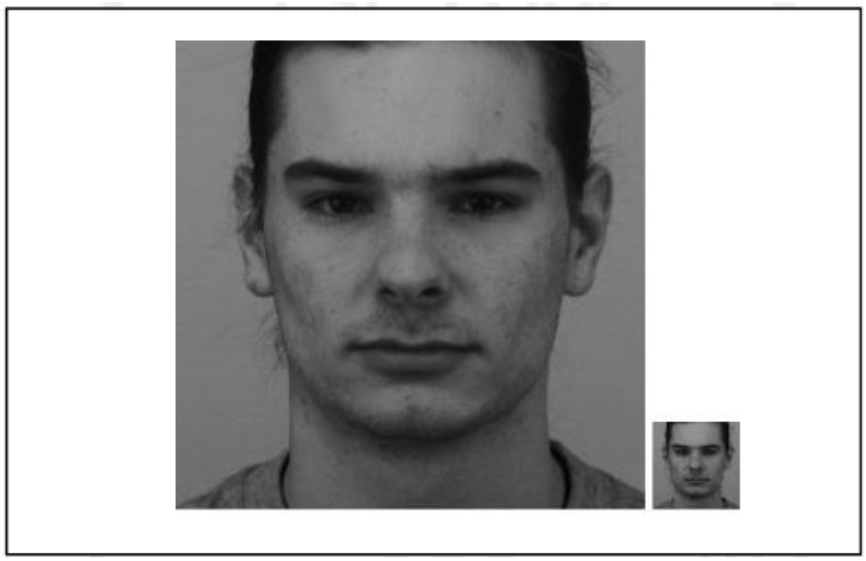
Gambar 4.2 Kode untuk mengubah ukuran data KDEF

Kode pada Gambar 4.2 dilakukan pertama-tama dengan menyimpan lokasi file yang akan digunakan berada, lokasi tersebut disimpan dengan *variable path.* Untuk penjelasan emosi pada folder dapat dilihat pada Tabel 4.2

Tabel 4.2 path folder emosi

Path folder	Emosi
0	Netral
1	Marah
2	Jijik
3	Takut
4	Senang
5	Sedih
6	Terkejut

Variabel width dan height memiliki nilai 48 yang bertujuan sebagai dimensi target setelah diubah. Penggunaan perulangan dilakukan sebanyak 7 kali, sesuai dengan jumlah folder emosi. Gambar 4.2 hanya menampilkan perulangan pada folder 0, namun perulangan serupa berlaku untuk folder lainnya, dengan perbedaan hanya pada path-nya. Proses pengubahan ukuran menggunakan fungsi *resize*(). Setelah dimensi berubah, gambar disimpan menggunakan fungsi *imsave*(). Hasil contoh perubahan ukuran dapat dilihat pada Gambar 4.3, dengan citra di sebelah kiri sebelum dilakukan perubahan ukuran dan citra di sebelah kanan setelah mengalami perubahan ukuran.



Gambar 4.1 perbedaan ukuran citra KDEF

4.3 Membagi data

Kode program pada Gambar 4.2 hingga Gambar 4.8 bertujuan untuk mengakses *data set yang* berupa gambar dengan format *PNG.* Pengaksesan data mengambil data dari *folder-folder* yang telah terpisah antara *folder* jenis *dataset* (data latih, data uji, dan data validasi) dan *folder* emosi. Penjelasan *folder* dapat dilihat pada-Tabel 4.1.

```
loc1 = 'E:/TA KU/Dataset/Gabungan/TRAIN/0'
loc2 = 'E:/TA KU/Dataset/Gabungan/TRAIN/1'
loc3 = 'E:/TA KU/Dataset/Gabungan/TRAIN/2'
loc4 = 'E:/TA KU/Dataset/Gabungan/TRAIN/3'
loc5 = 'E:/TA KU/Dataset/Gabungan/TRAIN/4'
loc6 = 'E:/TA KU/Dataset/Gabungan/TRAIN/5'
loc7 = 'E:/TA KU/Dataset/Gabungan/TRAIN/6'
```

Gambar 4.2 kode untuk menyimpan path dari lokasi data latih

Kode pada Gambar 4.4 mengambil data latih pada folder gabungan. Untuk penjelasan angka dari masing-masing folder dapat dilihat dari Tabel 4.1. Setelah

menyimpan *path* dari masing-masing folder emosi kemudian dilakukan pembacaan dan membuat label.

```
labels = []
for i in os.listdir(loc1):
labels.append(0)
for i in os.listdir(loc2):
labels.append(1)
for i in os.listdir(loc2):
labels.append(2)
for i in os.listdir(loc2):
labels.append(3)
for i in os.listdir(loc2):
labels.append(4)
for i in os.listdir(loc2):
labels.append(5)
for i in os.listdir(loc2):
labels.append(6)
features = []
from tqdm import tqdm
for i in tqdm(os.listdir(loc1)):
features.append(cv2.imread(os.path.join(loc1,i),0))
for i in tqdm(os.listdir(loc2)):
features.append(cv2.imread(os.path.join(loc2,i),0))
for i in tqdm(os.listdir(loc3)):
features.append(cv2.imread(os.path.join(loc3,i),0))
for i in tqdm(os.listdir(loc4)):
features.append(cv2.imread(os.path.join(loc4,i),0))
for i in tqdm(os.listdir(loc5)):
features.append(cv2.imread(os.path.join(loc5,i),0))
```

```
for i in tqdm(os.listdir(loc6)):
features.append(cv2.imread(os.path.join(loc6,i),0))
for i in tqdm(os.listdir(loc7)):
features.append(cv2.imread(os.path.join(loc7,i),0))
```

Gambar 4.3 Kode untuk membuat label dan membaca data latih

Pada Gambar 4.3, *variabel labels* digunakan untuk menghasilkan label sejumlah *folder* emosi yang ada di dalam data pelatihan. Fungsi *append* digunakan untuk menambahkan label sesuai dengan nama *folder. Variabel features* berperan dalam membaca data dari citra yang berformat jpg dengan menggunakan fungsi i*mread* dari *OpenCV.* Setelah pembacaan citra, data akan diubah menjadi *matriks* dan disimpan dalam *variabel features.*

```
loc1 = 'E:/TA KU/Dataset/Gabungan/TEST/GABUNGAN/0'
loc2 = 'E:/TA KU/Dataset/Gabungan/TEST/GABUNGAN/1'
loc3 = 'E:/TA KU/Dataset/Gabungan/TEST/GABUNGAN/2'
loc4 = 'E:/TA KU/Dataset/Gabungan/TEST/GABUNGAN/3'
loc5 = 'E:/TA KU/Dataset/Gabungan/TEST/GABUNGAN/4'
loc6 = 'E:/TA KU/Dataset/Gabungan/TEST/GABUNGAN/5'
loc7 = 'E:/TA KU/Dataset/Gabungan/TEST/GABUNGAN/6'
```

Gambar 4.4 kode untuk menyimpan path dari lokasi data uji gabungan

Kode pada gambar 4.4 menyimpan *Path* data uji pada folder gabungan. Untuk penjelasan angka pada *folder* dapat dilihat pada Tabel 4.1 setelah melakukann penyimpanan dilakukan hal yang sama dengan Gambar 4.5 dengan membuat label dan membaca data citra pada *folder* data uji.

```
loc1 = 'E:/TA KU/Dataset/Gabungan/VALIDATION/GABUNGAN/0'
loc2 = 'E:/TA KU/Dataset/Gabungan/VALIDATION/GABUNGAN/1'
loc3 = 'E:/TA KU/Dataset/Gabungan/VALIDATION/GABUNGAN/2'
```

```
loc4 = 'E:/TA KU/Dataset/Gabungan/VALIDATION/GABUNGAN/3'
loc5 = 'E:/TA KU/Dataset/Gabungan/VALIDATION/GABUNGAN/4'
loc6 = 'E:/TA KU/Dataset/Gabungan/VALIDATION/GABUNGAN/5'
loc7 = 'E:/TA KU/Dataset/Gabungan/VALIDATION/GABUNGAN/6'
```

Gambar 4.5 kode untuk menyimpan path dari lokasi data validasi

Kode pada Gambar 4.5 menyimpan *path* data validasi pada folder gabungan. Untuk penjelasan angka pada *folder* dapat dilihat pada Tabel 4.1. Setelah melakukan penyimpanan dilakukan hal yang sama dengan gambar 4.5 dengan membuat label dan membaca data citra pada *folder* data uji.

```
Loc1 = ‘E:/TA KU/Dataset/New folder/Gabungan/TEST/CK+/0’
loc2 = ‘E:/TA KU/Dataset/New folder/Gabungan/TEST/CK+/1’
loc3 = ‘E:/TA KU/Dataset/New folder/Gabungan/TEST/CK+/2’
loc4 = ‘E:/TA KU/Dataset/New folder/Gabungan/TEST/CK+/3’
loc5 = ‘E:/TA KU/Dataset/New folder/Gabungan/TEST/CK+/4’
loc6 = ‘E:/TA KU/Dataset/New folder/Gabungan/TEST/CK+/5’
loc7 = ‘E:/TA KU/Dataset/New folder/Gabungan/TEST/CK+/6’
```

Gambar 4.6 kode untuk menyimpan path dari lokasi data uji CK+

Kode pada Gambar 4.6 menyimpan *path* data uji CK+ pada folder gabungan. Untuk penjelasan angka pada *folder* dapat dilihat pada Tabel 4.1 setelah melakukan penyimpanan dilakukan hal yang sama dengan Gambar 4.5 dengan membuat label dan membaca citra pada *folder* data uji.

```
loc1 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/KDEF/0'
loc2 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/KDEF/1'
loc3 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/KDEF/2'
loc4 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/KDEF/3'
```

```
loc5 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/KDEF/4'
loc6 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/KDEF/5'
loc7 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/KDEF/6'
```

Gambar 4.7 kode untuk menyimpan path dari lokasi data uji KDEF

Kode pada gambar 4.7 menyimpan *path* data uji KDEF pada folder gabungan. Untuk penjelasan angka pada *folder* dapat dilihat pada Tabel 4.1. Setelah melakukan penyimpanan dilakukan hal yang sama dengan Gambar 4.5 dengan membuat label dan membaca citra pada folder data uji.

```
loc1 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/FER/0'
loc2 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/FER/1'
loc3 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/FER/2'
loc4 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/FER/3'
loc5 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/FER/4'
loc6 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/FER/5'
loc7 = 'E:/TA KU/Dataset/New folder/Gabungan/TEST/FER/6'
```

Gambar 4.8 Kode untuk menyimpan path dari lokasi data uji FER2013

Kode pada Gambar 4.8 menyimpan *path* data uji FER2013. Pada folder gabungan. Untuk penjelasan angka pada *folder* dapat dilihat pada Tabel 4.1. Setelah melakukan penyimpanan dilakukan hal yang sama dengan Gambar 4.5 dengan membuat label dan membaca data citra pada *folder* data uji.

```
train_data = pd.DataFrame()
train_data['emotion'] = labels
train_data['pixel_values'] = features


test_data = pd.DataFrame()
test_data['emotion'] = labels_test
test_data['pixel_values'] = test_features
```

```
val_data = pd.DataFrame()
val_data['emotion'] = labels_val
val_data['pixel_values'] = val_features

testck_data = pd.DataFrame()
testck_data['emotion'] = labels_testck
testck_data['pixel_values'] = testck_features

testkdef_data = pd.DataFrame()
testkdef_data['emotion'] = labels_testkdef
testkdef_data['pixel_values'] = testkdef_features

testfer_data = pd.DataFrame()
testfer_data['emotion'] = labels_testfer
testfer_data['pixel_values'] = testfer_features
```

Gambar 4.9 Kode untuk membuat data Frame

```
train_data.head()
```

	emotion	pixel_values
0	0	[[10, 2, 1, 2, 4, 3, 2, 7, 10, 12, 16, 14, 12,...
1	0	[[120, 100, 89, 100, 108, 100, 114, 144, 161, ...
2	0	[[83, 63, 55, 57, 66, 67, 69, 72, 80, 89, 91, ...
3	0	[[82, 72, 69, 65, 55, 60, 69, 92, 84, 75, 64, ...
4	0	[[220, 219, 220, 219, 202, 153, 64, 49, 42, 38...

Gambar 4.10 Hasil proses kode pada Gambar 4.9

Kode pada Gambar 4.11 membuat 6 data frame baru dengan fungsi *DataFrame()* yang terdapat pada *pandas.* Kemudian masing-masing *dataframe*

dibuat variabel baru dengan isi dari variabel label yang berisi label emosi dan feature yang berisikan matriks dari citra yang telah disimpan. Pada Gambar 4.12 dapat dilihat angka pada coloum yang berisi 0 merupakan label emosi 0 yaitu netral dan pada kolom pixel_values merupakan matriks dari data informasi citra.

```
#Bagian 1
features = np.array(features).reshape(-1,48,48,1)
test_features = np.array(test_features).reshape(-1,48,48,1)
val_features = np.array(val_features).reshape(-1,48,48,1)
testck_features = np.array(testck_features).reshape(-1,48,48,1)
testkdef_features = np.array(testkdef_features).reshape(-1,48,48,1)
testfer_features = np.array(testfer_features).reshape(-1,48,48,1)

#Bagian 2
features = features/255
test_features = test_features/255
val_features = val_features/255
testck_features = testck_features/255
testkdef_features = testkdef_features/255
testfer_features = testfer_features/255
#Bagian 3
labels = np_utils.to_categorical(labels)
labels_test =np_utils.to_categorical(labels_test)
labels_val =np_utils.to_categorical(labels_val)
labels_testck =np_utils.to_categorical(labels_testck)
labels_testkdef =np_utils.to_categorical(labels_testkdef)
labels_testfer =np_utils.to_categorical(labels_testfer)
```

Gambar 4.11 Kode untuk mengubah array data

```
features
```

```
[array([[10,  2,  1, ..., 14, 12, 46],
        [ 4,  1,  1, ..., 14, 11, 30],
        [ 1,  0,  0, ...,  5, 10, 24],
        ...,
        [65, 67, 69, ..., 77, 76, 75],
        [66, 66, 68, ..., 76, 76, 75],
        [68, 68, 68, ..., 76, 76, 76]], dtype=uint8),
```

Gambar 4.12 isi variabel features sebelum

```
features
```

```
array([[[[0.03921569],
         [0.00784314],
         [0.00392157],
         ...,
         [0.05490196],
         [0.04705882],
         [0.18039216]],

        [[0.01568627],
         [0.00392157],
         [0.00392157],
         ...,
         [0.05490196],
         [0.04313725],
         [0.11764706]],
```

Gambar 4.13 Isi variabel features sesudah

```
print(labels)
```

```
[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, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

Gambar 4.14 isi variabel labels sebelum

```
labels

array([[1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.]], dtype=float32)
```

Gambar 4.15 Isi variabel label sesudah

Kode pada Gambar 4.12 bagian 1 mengubah isi dari array yang ada dengan dimensi 48x48 dan nilai 1, mengingat gambar yang digunakan merupakan citra grayscale. Bagian 2 membagi isi yang terdapat pada variabel features dengan 256. Sedangkan bagian 3 mengubah isi dari variabel label dari tipe data integer menjadi matriks integer yang memiliki nilai biner, karena akan digunakan dalam pendeteksian emosi pada citra. Gambar 4.13 dan Gambar 4.15 menunjukkan isi dari variabel sebelum menerapkan kode pada Gambar 4.12, sedangkan Gambar 4.14 dan Gambar 4.16 menunjukkan isi variabel setelah penerapan kode tersebut.

4.3.1 Membangun arsitektur model

Kode pada gambar 4.16 digunakan untuk membangun sebuah arsitektur model yang dilakukan di menggunakan aplikasi Jupyter Notebook

```
model = Sequential()

#Layer 1

model.add(Conv2D(64,(3,3),padding = 'same',input_shape = (48,48,1)))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Dropout(0.25))


#Layer 2
```

```
model.add(Conv2D(128,(5,5),padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout (0.25))

#Layer 3
model.add(Conv2D(512,(3,3),padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout (0.25))

#Layer 4
model.add(Conv2D(512,(3,3), padding=’same’))
model.add(BatchNormalization())
model.add(Activation(‘relu’))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation(‘relu’))
model.add(Dropout(0.25))
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation(‘relu’))
model.add(Dropout(0.25))
```

```
model.add(Dense(7, activation='softmax'))
opt = Adam(lr = 0.0001)
model.compile(optimizer=opt,loss='categorical_crossentropy',
metrics=['accuracy'])
model.summary()
```

Gambar 4.16 Kode untuk membangun arsitektur CNN

Kode pada Gambar 4.17 membuat arsitektur model CNN. Model dibuat dengan fungsi *sequintal()* sehingga menghasilkan model yang berlapis-lapis. Untuk menambahkan layer menggunakan fungsi *add().* Arsitektur terdiri dari layer konvolusi,*flatten* dan *fully connected.*

Layer 1 menggunakan konvolusi dengan filter sebanyak 64. Ukuran *kernel* yang digunakan pada layer 1 yaitu 3x3. *Padding* yang digunakan yaitu *same. Layer* 1 terdapat juga *batch* normalisasi, *dropout* sebesar 0,25, dan *maxpooling* dengan ukuran 2x2. Aktivasi yang digunakan pada layer 1 yaitu menggunakan *ReLu.*

Layer 2 sampai dengan 4 tidak berbeda jauh dengan *layer 1.* Pada *layer 2 konvolusi* menggunakan filter sebanyak 128 dan menggunakan *kernel* denngan ukuran 5x5. *Layer 3* dan memiliki kesamaan pada konvolusi dengan filter sebanayak 512 dengan ukuran *kernel* sebesar 3x3. Setelah *layer* konvolusi terdapat *layer fully connected.*

Pada *layer fully connected* diawali dengan *flatten.* Kemudian 2 kali dilakukan dengan *dense, batch normalization, dropout* sebesar 0,25 dan aktivasi *ReLu.* Terakhir *fully connected* dengan aktivasi *softmax.*

Pada arsitektur, optimasi yang diterapkan menggunakan metode Adam dengan learning rate sebesar 0,0001. Selain itu, dibuat dua arsitektur tambahan dengan optimasi yang berbeda, yaitu RMS dan SGD, dengan pengaturan learning rate yang sama. Proses penyusunan model dilakukan melalui fungsi compile(). Fungsi loss yang digunakan adalah categorical cross entropy. Untuk mengevaluasi

hasil dari model yang telah dibangun, digunakan fungsi summary(). Rincian arsitektur yang telah dibuat dapat dilihat pada Gambar 4.12.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 48, 48, 64)	640
batch_normalization_1	(Batch (None, 48, 48, 64)	256
activation_1 (Activation)	(None, 48, 48, 64)	0
max_pooling2d_1 (MaxPooling2	(None, 24, 24, 64)	0
dropout_1 (Dropout)	(None, 24, 24, 64)	0
conv2d_2 (Conv2D)	(None, 24, 24, 128)	204928
batch_normalization_2 (Batch	(None, 24, 24, 128)	512
activation_2 (Activation)	(None, 24, 24, 128)	0
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 128)	0
dropout_2 (Dropout)	(None, 12, 12, 128)	0

conv2d_3 (Conv2D)	(None, 12, 12, 512)	590336
batch_normalization_3 (Batch	(None, 12, 12, 512)	2048
activation_3 (Activation)	(None, 12, 12, 512)	0
max_pooling2d_3 (MaxPooling2	(None, 6, 6, 512)	0
dropout_3 (Dropout)	(None, 6, 6, 512)	0
conv2d_4 (Conv2D)	(None, 6, 6, 512)	2359808
batch_normalization_4 (Batch	(None, 6, 6, 512)	2048
activation_4 (Activation)	(None, 6, 6, 512)	0
max_pooling2d_4 (MaxPooling2	(None, 3, 3, 512)	0
dropout_4 (Dropout)	(None, 3, 3, 512)	0
flatten_1 (Flatten)	(None, 4608)	0
dense_1 (Dense)	(None, 256)	1179904
batch_normalization_5 (Batch	(None, 256)	1024
activation_5 (Activation)	(None, 256)	0

dropout_5 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 512)	131584
batch_normalization_6 (Batch	(None, 512)	2048
activation_6 (Activation)	(None, 512)	0
dropout_6 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 7)	3591

Total params: 4,478,727
Trainable params: 4,474,759
Non-trainable params: 3,968

Gambar 4.17 Arsitektur model

Dari Gambar 4.18 diketahui parameter yang terdapat pada arsitektur tersebut berjumlah 4.478.727. Untuk parameter yang dapat pelatihan 4.474.759 dan parameter tidak dapat dilatih berjumlah 3.968.

4.3.2 Melatih Model

Pelatih model dilakukan setelah membuat model CNN. Pelatihan ini menggunakan *epoch* sebanyak 200. Berikut baris code dalam melakukan pelatihan.

```
epochs = 200
history = model.fit(x=features,
                    y=labels,
```

```
batch_size=64,
epochs=epochs,
validation_data = (val_features,labels_val)
)
```

Gambar 4.18 Kode pelatihan model

Baris kode pada Gambar 4.15, untuk menjalankan pelatiahan model digunakan fungsi *fit()*. Perkembangan pelatihan model disimpan dalam *variable history.* Variabel x dan y merujuk pada data yang akan digunakan dalam pelatihan. Variabel x merujuk pada isi matriks data pelatihan dan y merujuk pada label dari data pelatihan. Pelatiahan ini menggunakan *batch_size* 64 dan *epoch* sebanyak 200. *Validation_data* merujuk pada data validasi yang digunakan pada pelatihan model ini. Pada pelatihan ini dapat diketahui perkembangan *val_accuracy, accuracy, val_loss,* dan loss. Berikut tabel *progres* pelatihan model per 10 *epoch.*

Tabel 4.3 Progres Pelatihan dengan optimasi ADAM

Epoch	Loss	Akurasi	Validate loss	Validate akurasi
1	2,2896	0,1584	1,9662	0,1429
10	1,8246	0,2973	2,1075	0,1535
20	1,0542	0,4164	1,7153	0,3731
30	1,9293	0,4935	1,5432	0,4563
40	1,1363	0,5749	1,4971	0,4797
50	0,9505	0,6364	1,4613	0,4925
60	0,7659	0,7193	1,4654	0,5032
70	0,606	0,7845	1,4536	0,5458
80	0,4729	0,8284	1,7317	0,5117
90	0,3534	0,8727	0,7360	0,5394
100	0,282	0,8975	1,8288	0,5139
110	0,2231	0,925	1,934	0,5352
120	0,184	0,9342	2,0181	0,533
130	0,155	0,4977	2,0439	0,5394
140	0,1286	0,9604	2,128	0,4309
150	0,1067	0,963	2,2798	0,5458
160	0,101	0,9667	2,281	0,5245
170	0,0896	0,9699	2,2833	0,5565
180	0,0769	0,792	2,5984	0,5352
190	0,0748	0,9762	2,5019	0,5267

200	0,0559	0,9831	2,4956	0,5501

Pelatihan model menggunkan data sebanyak 3738 data training dan 469 data validasi. Pelatihan menghabiskan waktu 1 hari 18 jam 12 menit 54 detik.

4.3.3 Melakukan Pengujian Model

Pengujian dilakukan menggunakan empat set data, yaitu CK+, FER2013, KDEF, dan gabungan dari ketiga *dataset* tersebut. Pada percobaan pertama, *eksperimen* dilakukan dengan memvariasikan *optimasi* yang digunakan. Terdapat tiga jenis *optimasi* yang diuji, yaitu Adam, SGD, dan RMS. Semua pengujian dilakukan dengan konfigurasi yang serupa, dengan *epoch* setinggi 200, ukuran batch 64, dan *learning rate* sebesar 0,0001.

Tabel 4.4 Perbandingan hasil dengan optimasi berbeda

Dataset	akurasi		
	Adam	SGD	RMS
FER	0,52	0,19	0,48
KDEF	0,82	0,14	0,75
CK+	0,77	0,14	0,71
Gabungan	0,57	0,19	0,53

Dari Tabel 4.4 Nilai terbaik pada optimasi Adam. Pada data uji KDEF memiliki nilai paling tinggi hingga 0,81. Selanjutnya menguji dengan optimasi terbaik yaitu adam dengan perbedaan *epoch* saat melakukan pelatihan.

Tabel 4.5 Perbandingan hasil dengan epoch yang berbeda

Dataset	***Epoch***	
	200	**200**
FER	0,52	0,49
KDEF	0,81	0,77
CK+	0,77	0,71
Gabungan	0,57	0,54

4.3.4 Perbandingan hasil

Dalam pengujian model, akurasi tertinggi terlihat pada penggunaan *optimasi* Adam. Pengujian dilakukan dengan empat *dataset* yang berbeda, yaitu FER2013, CK+, KDEF, dan gabungan dari ketiga *dataset* tersebut. Untuk perbandingan hasil, akan

diambil *confusion matrix* dari setiap dataset. Hasil pengujian ditampilkan dalam bentuk *Confusion Matrix*. Berikut adalah baris kode untuk pengujian dengan *dataset* yang digabungkan.

```
test_true = np.argmax(labels_test, axis=1)
test_pred = np.argmax(model.predict(test_features), axis=1)
print(metrics.confusion_matrix(test_true, test_pred,
labels=[0,1,2,3,4,5,6]))
print(metrics.classification_report(test_true, test_pred,
labels=[0,1,2,3,4,5,6]))
```

Gambar 4.19 kode untuk melakukan pengujian dengan data uji gabungan

Kode pada gambar 4.14 fungsi *argmax()* digunakan untuk mendapatkan indeks nilai tertinggi disepanjang sumbu. *Test_true* menyimpan isi dari variabel labels_test. Fungsi *predict().* Digunakan untuk melakukan prediksi dengan model yang telah terlatih. Test_pred berisi hasil prediksi dari variabel test_features. Fungsi *confusion_matrix* digunakan untuk mengevaluasi keakuratan klasifikasi. Fungsi *classification_report*() untuk membangun laporan teks yang menunjukan metrik klasifikasi utama. Pada fungsi tersebut terdapat *recall*,presisi,dan ukuran F.

Tabel 4.6 Confussion matrix dengan optimasi ADAM data uji gabungan

		prediksi					
		0	1	2	3	4	5
Aktual	0	35	8	4	5	7	7
	1	12	31	3	3	4	8
	2	4	3	50	1	3	6
	3	8	13	4	21	4	10
	4	5	1	1	1	55	4
	5	16	10	2	5	6	27
	6	3	1	1	7	1	3

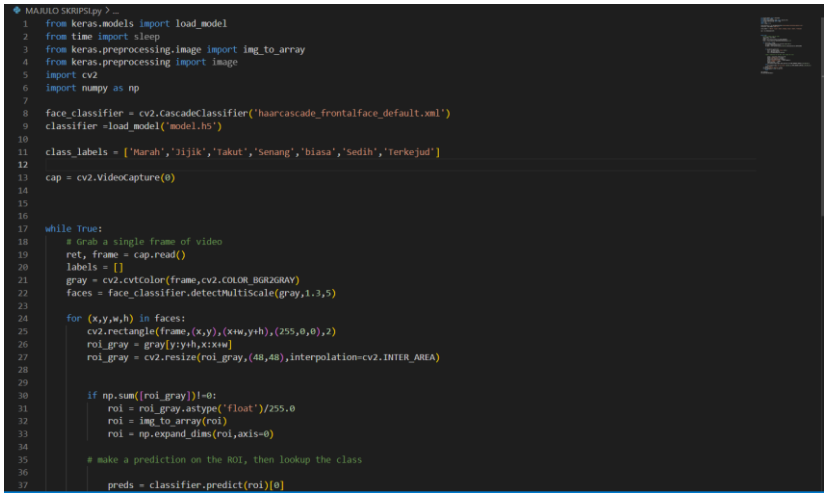
4.4 Hasil Rancangan Program

Hasil dari program yang telah dikembangkan mencerminkan *implementasi* dari perancangan menjadi sebuah aplikasi yang masih dapat dikembangkan lebih lanjut untuk perbaikan dan peningkatan akurasi. Tingkat akurasi yang diambil dari *dataset*

ekspresi wajah dapat ditingkatkan melalui berbagai cara, seperti penambahan variasi ekspresi pada *dataset* atau melakukan pelatihan ulang pada model untuk mencapai tingkat akurasi yang lebih sesuai dengan ekspresi wajah pengguna.

4.5 Tampilan Halaman Program

Berikut merupakan tampilan halaman program ketika program akan dijalankan. Tampilan Halaman program dapat dilihat pada gambar 4.20.



```
MAJULO SKRIPS.py > ...
from keras.models import load_model
from time import sleep
from keras.preprocessing.image import img_to_array
from keras.preprocessing import image
import cv2
import numpy as np

face_classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
classifier =load_model('model.h5')

class_labels = ['Marah','Jijik','Takut','Senang','biasa','Sedih','Terkejud']

cap = cv2.VideoCapture(0)



while True:
    # Grab a single frame of video
    ret, frame = cap.read()
    labels = []
    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(gray,1.3,5)

    for (x,y,w,h) in faces:
        cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h,x:x+w]
        roi_gray = cv2.resize(roi_gray,(48,48),interpolation=cv2.INTER_AREA)


        if np.sum([roi_gray])!=0:
            roi = roi_gray.astype('float')/255.0
            roi = img_to_array(roi)
            roi = np.expand_dims(roi,axis=0)

        # make a prediction on the ROI, then lookup the class

            preds = classifier.predict(roi)[0]
```

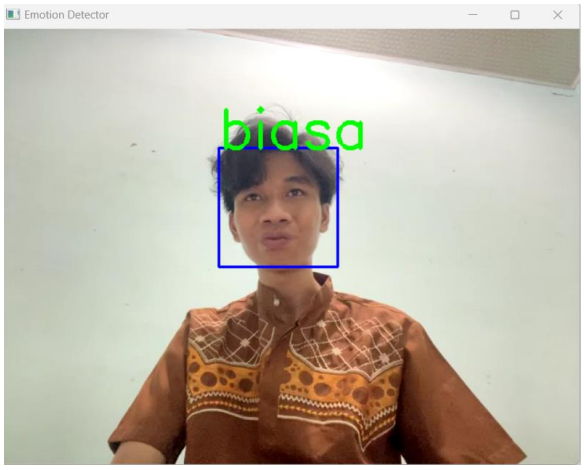
Gambar 4.20 run debug aplikasi deteksi wajah

4.6 Hasil

Dari uji coba aplikasi ini, diperoleh berbagai hasil ekspresi wajah yang mencakup 7 kategori ekspresi dari beberapa dataset yang digunakan dalam program ini. Langkah-langkah ini diimplementasikan untuk meningkatkan akurasi keseluruhan program.

4.6.1 Testing Ekspresi Biasa

Pada gambar 4.21 dibawah menunjukan bahwa ekspresi wajah biasa (netral) berhasil dikenali. Hasil tersebut didapat dengan mengambil gambar layar (print screen) pada saat aplikasi deteksi wajah berjalan.



Gambar 4.21 deteksi ekspresi biasa (netral)

4.6.2 Tesiting ekspresi senang

Pada gambar 4.22 dibawah menunjukan bahwa ekspresi wajah senang berhasil dikenali. Hasil tersebut didapat dengan mengambil gambar layar (*print screen*) pada saat aplikasi deteksi wajah berjalan.

Gambar 4.22 deteksi ekspresi senang

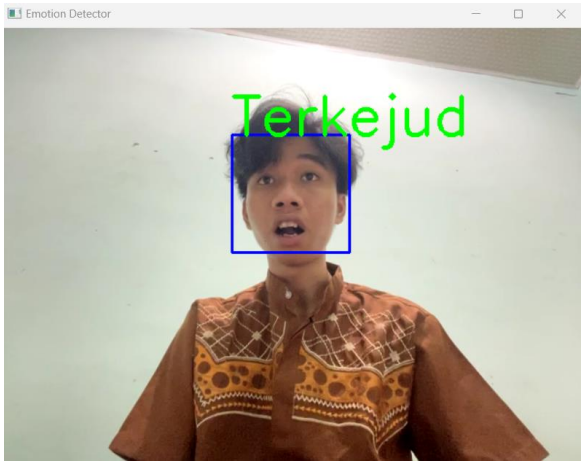
4.6.3 Testing ekspresi sedih

Pada gambar 4.23 dibawah menunjukan bahwa ekspresi wajah sedih berhasil dikenali. Hasil tersebut didapat dengan mengambil gambar layar (*print screen*). Pada saat aplikasi deteksi wajah berjalan

Gambar 4.23 deteksi ekspresi sedih

4.6.4 Testing ekspresi terkejut

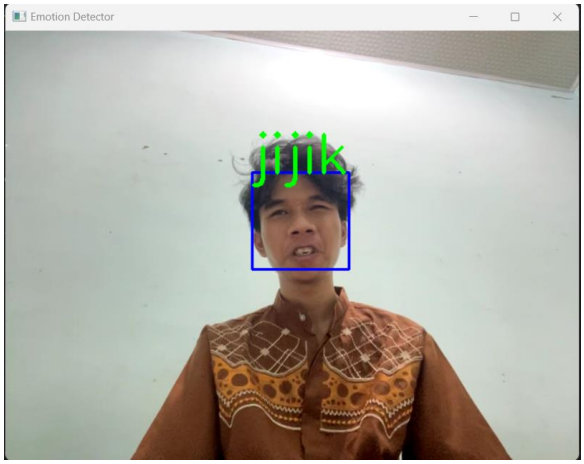
Pada gambar 4.24 dibawah menunjukan bahwa ekspresi wajah terkejut berhasil dikenali. Hasil tersebut didapat dengan mengambil gambar layar (*print screen*). Pada saat aplikasi deteksi wajah berjalan.



Gambar 4.24 Deteksi ekspresi terkejut

4.6.5 Testing ekspresi Jijik

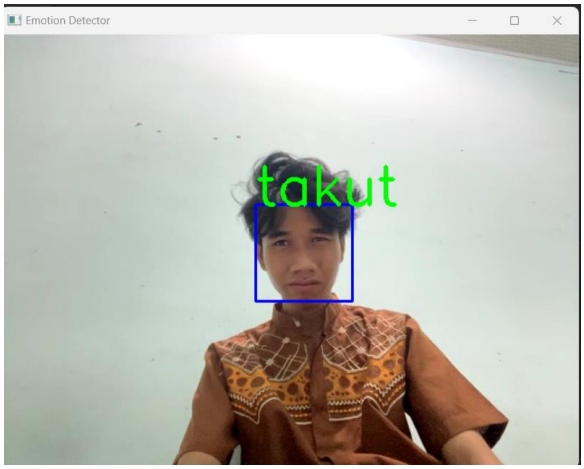
Pada gambar 4.25 dibawah menunjukan bahwa ekspresi wajah jijik berhasil dikenali. Hasil tersebut didapat dengan mengambil gambar layar (*print screen*). Pada saat aplikasi deteksi wajah berjalan.



Gambar 4.25 Deteksi ekspresi jijik

4.6.6 Testing ekspresi Takut

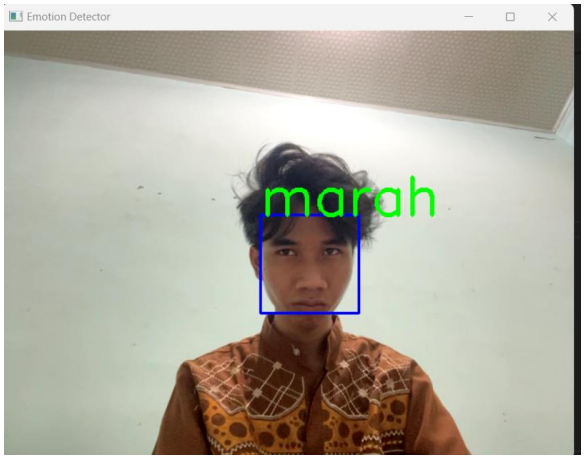
Pada gambar 4.26 dibawah menunjukan bahwa ekspresi wajah takut berhasil dikenali. Hasil tersebut didapat dengan mengambil gambar layar (*print screen*). Pada saat aplikasi deteksi wajah berjalan.



gambar 4.26 Deteksi ekspresi takut

4.6.7 Testing ekspresi marah

Pada gambar 4.27 dibawah menunjukan bahwa ekspresi wajah marah berhasil dikenali. Hasil tersebut didapat dengan mengambil gambar layar (*print screen*). Pada saat aplikasi deteksi wajah berjalan.



gambar 4.27 Deteksi ekspresi marah