BAB IV
HASIL DAN PEMBAHASAN

Pada bab ini menjelaskan tahapan-tahapan penelitian secara rinci proses yang dilakukan dan hasil penelitian

4.1 Pra-Poses Data

Sebelum melakukan penelitian ada beberapa praproses yang harus disiapkan meliputi input dataset, *split* data, resize citra dan augmentasi data

4.1.1 Input Dataset

Pada proses input dataset gambar yang telah disimpan ke dalam *Google Drive*, kemudian dilakukan proses penginputan dataset ke dalam program yang telah dibuat menggunakan *platform Google Colab* berbayar dengan perintah "MyDrive/DATASET_MATA_IKAN_2" dan di setiap gambarnya memiliki folder tersendiri sesuai kelasnya yaitu segar dan tidak segar. *Syntax input* dataset seperti gambar 4.1 *Input* Data Gambar.

```
from google.colab import drive
drive.mount('/content/drive')
data_dir = '/content/drive/MyDrive/DATASET_MATA_IKAN_2/MATA_IKAN_1.1/'
```

Gambar 4.1 Input Data Gambar

4.1.2 Pembagian Data

Pada *split* data atau pembagian data pada penelitian ini menggunakan fungsi yang ada di "sckit-learn", karena fungsi yang digunakan adalah fungsi "train_test_split" berikut adalah gambar program dari *split* data.

```
df_train_val, df_test = train_test_split(df, random_state=666, test_size=0.1 )
df_train_val.to_csv('/content/drive/MyDrive/DATASET_MATA_IKAN_2/train_label.csv', index=False)
df_test.to_csv('/content/drive/MyDrive/DATASET_MATA_IKAN_2/test_label.csv', index=False)
image_size=(224, 224)
```

Gambar 4.2 Split Data

Kode program di atas merupakan kode *split* data dengan perbandingan 9:1, dimana data *train* 90% dan data *testing* 10%.

4.1.3 Resize Citra dan Augmentasi Data

Pada proses *resize citra* peneliti menggunakan *resize* dengan target 244x244 *pixel* dimana proses *resize* ukuran citra dari ukuran asli yang

didapat citra berukuran 3040x3040 *pixel* diproses menjadi 244x244 *pixel*. Proses augmentasi data menggunakan *on the fly* atau *real time* dimana proses ini merupakan fungsi yang ada di "tensorflow" yaitu "ImageDataGenerator".

```
save_to = '/content/drive/MyDrive/DATASET_MATA_IKAN_2/MATA_IKAN_1.1/segar1.1/'
target_size = (224,224)
train_datagen = ImageDataGenerator(
                  rotation_range = 30,
                  horizontal_flip = True,
                   zoom_range = 0.3,
```

Gambar 4.3 Augmentasi Data

Program "ImageDataGenerator" pada gambar diatas berfungsi mengaugmentasi data secara *real time* atau *on the fly*, dengan menormalisasi *citra, rotation range* = 30, *Horizontal_flip = True dan zoom_range* = 30.

4.2 Pembangunan Model

Pada proses pembangunan model ini, peneliti mengimplementasikan atau menggunakan fungsi dari *keras Functional API* dimana fungsi ini telah disediakan oleh tensorflow, dan selanjutnya terhubung dengan *layer, layer convolutional layer, batch normalization, max pooling layer, fully connected layer* dan *dropout* seperti pada gambar 4.5 di bawah ini.

```
model = keras.Sequential()

# Creating first block- (2 Convolution + 1 Max pool)
model.add(Conv2D(filters= 32, kernel_size= (3,3), strides= (1,1), padding='same', input_shape= (224, 224, 3), activation= 'relu'))
# model.add(BatchNormalization())
model.add(Conv2D(filters= 32, kernel_size= (3,3), strides= (1,1), padding='same', activation= 'relu'))
# model.add(BatchNormalization())
model.add(MaxPool2D(pool_size= (2,2), strides=(2,2)))

# Creating second block- (2 Convolution + 1 Max pool)
model.add(Conv2D(filters= 64, kernel_size= (3,3), strides= (1,1), padding='same', activation= 'relu'))
# model.add(BatchNormalization())
model.add(Conv2D(filters= 64, kernel_size= (3,3), strides= (1,1), padding='same', activation= 'relu'))
# model.add(BatchNormalization())
model.add(MaxPool2D(pool_size= (2,2), strides=(2,2)))

# Creating third block- (3 Convolution + 1 Max pool)
model.add(Conv2D(filters= 128, kernel_size= (3,3), strides= (1,1), padding='same', activation= 'relu'))
# model.add(BatchNormalization())
model.add(Conv2D(filters= 128, kernel_size= (3,3), strides= (1,1), padding='same', activation= 'relu'))
# model.add(BatchNormalization())
model.add(Conv2D(filters= 128, kernel_size= (3,3), strides= (1,1), padding='same', activation= 'relu'))
# model.add(BatchNormalization())
model.add(MaxPool2D(pool_size= (2,2), strides=(2,2)))

# Creating fourth block- (3 Convolution + 1 Max pool)
model.add(Conv2D(filters= 256, kernel_size= (3,3), strides= (1,1), padding='same', activation= 'relu'))
# model.add(BatchNormalization())
model.add(Conv2D(filters= 256, kernel_size= (3,3), strides= (1,1), padding='same', activation= 'relu'))
# model.add(BatchNormalization())
model.add(Conv2D(filters= 256, kernel_size= (3,3), strides= (1,1), padding='same', activation= 'relu'))
# model.add(BatchNormalization())
model.add(MaxPool2D(pool_size= (2,2), strides=(2,2)))

# Creating fifth block- (3 Convolution + 1 Max pool)
model.add(Conv2D(filters= 512, kernel_size= (3,3), strides= (1,1), padding='same', activation= 'relu'))
# model.add(BatchNormalization())
model.add(Conv2D(filters= 512, kernel_size= (3,3), strides= (1,1), padding='same', activation= 'relu'))
# model.add(BatchNormalization())
model.add(Conv2D(filters= 512, kernel_size= (3,3), strides= (1,1), padding='same', activation= 'relu'))
# model.add(BatchNormalization())
model.add(MaxPool2D(pool_size= (2,2), strides=(2,2)))

# Flattening the pooled image pixels
model.add(Flatten())

# Creating 2 Dense Layers
model.add(Dense(units= 4096, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(units= 4096, activation='relu'))
model.add(Dropout(rate=0.5))
# Creating an output layer
model.add(Dense(units= 2, activation='softmax'))
```

Gambar 4.4 Pembangunan Model VGG-16

4.3 Pelatihan Model

Pada tahap pelatihan model ini, peneliti menggunakan beberapa parameter untuk memproses data seperti jenis *optimizer* yang digunakan adalah SGD untuk optimasinya dimana jumlah *epoch = 100*, *learning rate* = 0,003, untuk data yang digunakan pelatihan model adalah data *training.* Di bawah gambar 4.6 adalah implementasi dari *code* program pelatihan model.

```
k = 10
kf =  StratifiedKFold(n_splits = k, shuffle = True, random_state = 666)
i = 1
for train_index, val_index in kf.split(df_train_val['path'], df_train_val['label']):
  if i == 1:
    print('Train shape:',df_train_val.iloc[train_index].shape)
    print('val shaoe:', df_train_val.iloc[val_index].shape)

    csv_filepath = CSVLogger('/content/drive/MyDrive/DATASET_MATA_IKAN_2/' +"Model/" + '_fold_' + str(i) + '_model.log')

    model_checkpoint_callback = ModelCheckpoint(
        filepath='/content/drive/MyDrive/DATASET_MATA_IKAN_2/' + "Model/" + '_fold_' + str(i) + '_model.h5',
        save_weights_only=False,
        monitor='val_accuracy',
        mode='max',
        verbose=1,
        save_best_only=True)

    csvlogger = csv_filepath
    callbacks = []
    callbacks.append(csvlogger)
    callbacks.append(model_checkpoint_callback)

    train_generator = datagen.flow_from_dataframe(
        dataframe=df_train_val.iloc[train_index],
        x_col="path",
        y_col="label",
        shuffle=True,
        target_size=(224,224),
        class_mode="categorical",)

    val_generator = datagen.flow_from_dataframe(
        dataframe=df_train_val.iloc[val_index],
        x_col="path",
        y_col="label",
        shuffle=False,
        target_size=(224,224),
        class_mode="categorical",)
    optimizer = SGD(learning_rate=0.003,momentum=0.9)

    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

    result = model.fit(x=train_generator,validation_data=val_generator, epochs=100, callbacks=callbacks )
    break
  i += 1
```

Gambar 4.5 Pelatihan Model

4.4 Pengujian Model

Pada tahap proses pengujian model ini peneliti melakukan proses pengujian model yang telah dirancang dan dibangun, dari terbentuknya pembangunan model ini bertujuan untuk menunjukan bahwasannya model yang telah dibangun dapat diharapkan dan memenuhi hasil akurasi, *sensifty*, dan *specificity*.

4.4.1 Hasil Pengujian K-Fold Cross Validation

Pada tahap pengujian ini dilakukan menggunakan *K-Fold cross validation*, dimana pada pengujian ini peneliti membagi ke dalam dua *folder* yang terdapat di dalamnya 1920 data citra dan dibagi menjadi data *train* dan data *validation* yang diambil secara acak, secara keseluruhan data tersebut terbagi menjadi data *train* sebanyak 1728 citra dan data *validation* 192 citra. Dapat dilihat pada Tabel 4.1 hasil dari pengujian *K-Fold Cros Validation.*

Tabel 4.1 Hasil Pengujian Menggunakan *K-Fold Validation*

K-Fold cross validation, K=10	Akurasi Keseluruhan
Fold 1	75,1
Fold 2	100
Fold 3	72,8
Fold 4	99,4
Fold 5	70,5
Fold 6	85,5
Fold 7	97,6
Fold 8	97,1
Fold 9	68,6
Fold 10	97,6

Dari hasil pengujian model dapat dilihat pada tabel 4.1 hasil Pengujian menggunakan *K-Fold Validation* dimana dari 10 *fold* yang telah diuji mendapatkan satu *fold* yang memiliki nilai akurasi sangat tinggi yaitu pada *fold* 2 mendapatkan akurasi sebesar 100%, sedangakan pada *fold* 5 mendapatkan nilai akurasi paling rendah dari *fold* lainnya yaitu sebesar 70,5%.

Tabel 4.2 *Confusion Matrix*

		Predict	
		Segar	**Tidak Segar**
Actual	**Segar**	**87**	**0**
	Tidak Segar	**2**	**83**

Hasil dari pengujian model pada tabel 4.2 *Confusion Matrix* mendapatkan hasil prediksi dari total citra latih 1728 citra dibagi secara acak ke 10 *fold* pada *fold* 2 mendapatkan 172 citra.

Tabel 4.3 Akurasi, Sensitivitas, dan Spesifisitas

Kelas	Akurasi (%)	Sensitivitas (%)	Spesifitas (%)
Segar	0,98844	1,0000	0,97647
Tidak Segar	0,98844	0,97647	1,0000

Pada *fold* 2 ini performa yang diperoleh sangatlah baik, dapat dilihat bahwa kinerja pada setiap kelasnya, segar dan tidak segar mendapatkan nilai akurasi, sensitivitas dan spesifisitas yang memiliki nilai yang sangat tinggi dan bisa dikatakan mampu memprediksi kelas segar dan tidak segar dengan sangat baik. Sebagai hasilnya bisa dilihat pada tabel 4.3 Akurasi, *Sensitivitas*, dan *Spesifisitas*.

4.4.2 Uji Coba Efektivitas Model

Pada skenario uji coba efektivitas ini menguji model yang telah dibentuk apakah mengalami *overfitting* atau *underfitting*, model yang digunakan adalah *fold* 2 dimana pada model ini mendapatkan nilai akurasi yang sangat tinggi dibandingkan dengan nilai fold lainnya. Kemudian data yang digunakaan untuk uji coba efektivitas model ini adalah data *citra* uji yang telah dibagi di pembagian data yaitu 9:1 atau 10% dari total 1920 *citra* yaitu sebanyak 192 *citra*, yang sudah mewakili kelas mata ikan segar dan tidak segar

4.4.2.1 Uji Coba Efektivitas Model

Pada performa dari model *fold* 2 ini dapat dilihat pada tabel 4.1 mendapatkan nilai akurasi yang sangat baik dikarenakan mendapatkan nilai yang sangat tinggi yaitu 100%, dan dilihat dari nilai spesifisitas pada setiap kelas model *fold* 2 mampu memprediksi kelas sangat baik.

Pada uji coba efektivitas model, *fold* 2 diuji kembali menggunakan citra data baru yang belum pernah digunakan sama sekali pada pelatihan maupun uji coba data *testing*. Tabel hasil pengujian-pengujian dapat dilihat di tabel 4.4 dan 4.5 di bawah ini.

Tabel 4.4 Confusion Matrix

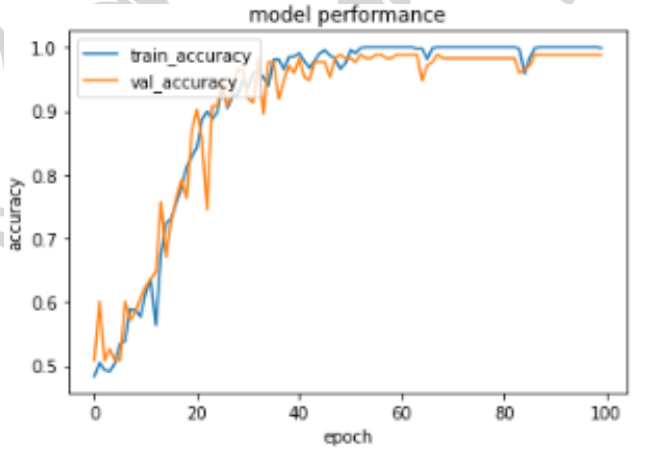
		Predict	
		Segar	**Tidak Segar**
Actual	**Segar**	**87**	**0**
	Tidak Segar	**3**	**102**

Tabel 4.5 Akurasi, Sensitivitas, dan Spesifisitas

Kelas	Akurasi (%)	Sensitivitas (%)	Spesifitas (%)
Segar	0,98438	1,0000	0,97143
Tidak Segar	0,98438	0,97143	1,0000

Hasil dari uji coba efektivitas model ini *fold* 2 mendapatkan nilai akurasi 98,4%, *sensitivitas* 98,5%, dan *spesifisitas* 98,5%. Dilihat dari performa setiap kelas terdapat kelas yang mendapatkan nilai *sensitivitas* lebih tinggi daripada nilai *spesifitas* begitu juga sebaliknya.

Sehingga dari pernyataan tersebut kesimpulannya adalah model *fold* 2 kurang baik efektivitasnya dalam mendeteksi kesegaran ikan, pada *fold* 2 ini mengalami *overfitting*, dikarenakan pada saat uji coba dengan dataset baru (citra mata ikan lemuru) mengalami perubahan atau penurunan kinerja dari saat pelatihan.



Grafik 4.1 Grafik Pelatihan