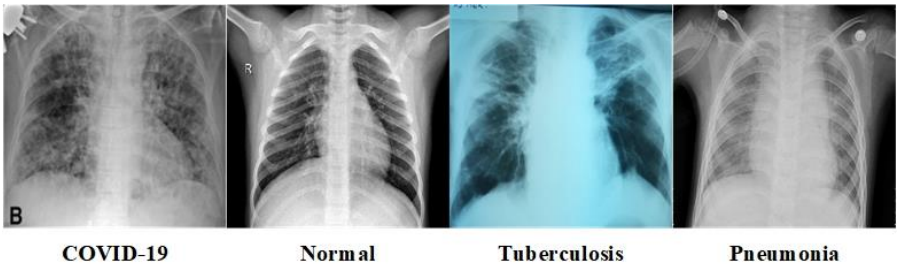
BAB IV
HASIL DAN PEMBAHASAN

4.1 Pengumpulan Data

Proses pengumpulan data dilakukan dengan cara mengunduh pada website dengan link https://www.kaggle.com/datasets/tawsifurrahman/tuberculosis-tb-chest-xray-dataset dan https://www.kaggle.com/datasets/tawsifurrahman/covid19-radiography-database. Hasil proses pengumpulan data ditunjukkan pada Tabel 4.1 dan citra rontgen paru ditunjukkan pada Gambar 4.1

Tabel 4.1 Hasil pengumpulan data

No.	Data	Kelas
1	Citra Rontgen Paru-Paru	COVID-19, Normal, Tuberculosis, Pneumonia



Gambar 4.1 Citra rontgen paru-paru

Hasil pengumpulan total data yang didapatkan adalah 15.884 citra yang dibagi menjadi 4 penyakit yaitu COVID-19, Normal, Tuberculosis, dan Pneumonia. Setiap kelas penyakit memiliki total citra rontgen yang berbeda.

4.1.1 Penyakit COVID-19

Pada data rontgen paru dengan penyakit COVID-19 total citra yang didapat adalah 3616 gambar. Citra dibagi menjadi *training*, *validation*, dan *testing* yaitu masing-masing mendapatkan persentase yang berbeda. Data *training* 80%, *validation* 10%, dan *testing* 10%.

4.1.2 Penyakit Normal

Pada data rontgen paru dengan penyakit Normal total citra yang didapat adalah 9573 gambar. Citra dibagi menjadi *training*, *validation*, dan *testing* yaitu masing-masing mendapatkan persentase yang berbeda. Data *training* 80%, *validation* 10%, dan *testing* 10%.

4.1.3 Penyakit Tuberculosis

Pada data rontgen paru dengan penyakit Tuberculosis total citra yang didapat adalah 700 gambar. Citra dibagi menjadi *training, validation*, dan *testing* yaitu masing-masing mendapatkan persentase yang berbeda. Data *training* 80%, *validation* 10%, dan *testing* 10%.

4.1.4 Penyakit Pneumonia

Pada data rontgen paru dengan penyakit Tuberculosis total citra yang didapat adalah 700 gambar. Citra dibagi menjadi *training, validation*, dan *testing* yaitu masing-masing mendapatkan persentase yang berbeda. Data *training* 80%, *validation* 10%, dan *testing* 10%.

4.2 Data Preprocessing

Data mentah yang telah terkumpul maka selanjutnya dilakukan tahap preprocessing yang bertujuan mempersiapkan data mentah menjadi data yang siap digunakan, beberapa proses yang dilakukan adalah sebagai berikut.

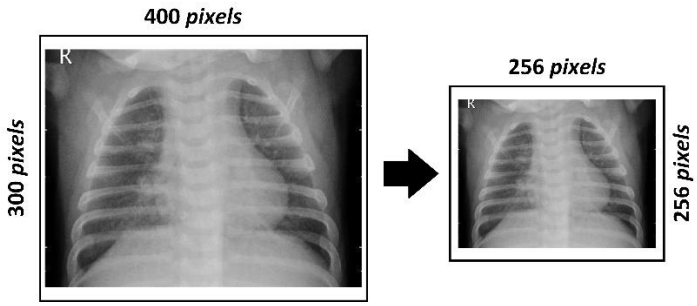
4.2.1 *Resize*

Proses *resize* dilakukan untuk menyamakan ukuran pada semua citra, semua data citra dilakukan *resize* dengan ukuran 256x256 pixel. Proses *resize* ditunjukkan pada Kode Program 4.1 dan hasil proses *resize* ditunjukkan pada Gambar 4.2.

```
import PIL
import os
from PIL import Image
```

```
f = r'/content/drive/MyDrive/coba_1/coba_2/'
for file in os.listdir(f):
    f_img = f+"/"+file
    img = Image.open(f_img)
    img = img.resize((256,256))
    img.save(f_img)
```

Kode Program 4.1 Proses *resize*



Gambar 4.2 Hasil *resize*

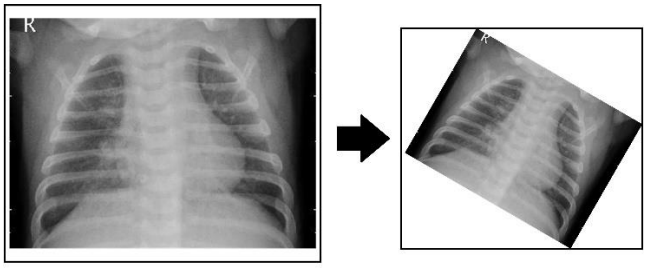
Pada proses *resize* data menggunakan *library* os dan pillow. Pada *variable* f digunakan untuk menunjukkan *directory* folder citra berada. Setiap citra dilakukan proses *resize*, dimulai dengan membuka citra dengan fungsi `Image.open` yang ditampung pada *variable* img, selanjutnya citra diatur ukuran lebar dan tinggi dengan fungsi `img.resize`, setelah itu citra akan disimpan dengan fungsi `img.save`.

4.2.2 Data *Augmentation*

Data yang telah diproses *resize* dan memiliki dimensi yang sama selanjutnya dilakukan proses augmentasi data, proses ini bertujuan untuk meningkatkan jumlah data yang dimiliki. Proses augmentasi data ditunjukkan pada Kode Program 4.2 dan hasil proses augmentasi data ditunjukkan pada Gambar 4.3.

```
train_datagen = ImageDataGenerator(rescale=1. / 255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   rotation_range=15,
                                   horizontal_flip=True,
                                   fill_mode='nearest')
val_datagen = ImageDataGenerator(rescale=1. / 255,
                                 shear_range=0.2,
                                 zoom_range=0.2,
                                 rotation_range=15,
                                 horizontal_flip=True,
                                 fill_mode='nearest')
test_datagen = ImageDataGenerator(rescale=1. / 255,
                                  shear_range=0.2,
                                  zoom_range=0.2,
                                  rotation_range=15,
                                  horizontal_flip=True,
                                  fill_mode='nearest')
```

Kode Program 4.2 Proses augmentasi data



(a) (b)

Gambar 4.3 Hasil augmentasi data

Pada proses augmentasi data dilakukan pada semua data *training, validation, testing*. Proses augmentasi menggunakan fungsi `ImageDataGenerator` pada *library* tensorflow dan menggunakan beberapa parameter seperti `rescale` bertujuan untuk menormalisasi citra dengan membagi *pixel* terkecil dengan *pixel* terbesar. `Shear_range` bertujuan untuk memutar citra dengan putaran sebanyak 0.2 derajat. `Zoom_range` bertujuan untuk memperbesar citra sebesar 0.2 dari luas citra. `Rotation_range` bertujuan untuk memutar citra dengan sudut 15 derajat secara acak. `Horizontal_flip` bertujuan membalik citra secara horizontal. `Fill_mode` bertujuan mengisi ruang kosong pada citra.

4.3 *Convolutional Neural Networks*

Data yang telah selesai dilakukan *preprocessing* selanjutnya akan diimplementasikan dengan metode CNN untuk mengetahui apakah metode CNN cocok digunakan pada klasifikasi penyakit paru atau tidak.

4.3.1 Instalisasi Library

Pada tahap ini beberapa *library* akan digunakan agar dapat melaksanakan proses *modeling* dengan lebih mudah. *Library* bantuan yang digunakan pada pengimplementasian ini ditunjukkan pada Kode Program 4.3.

```
import os
import pandas as pd
import math
import keras
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.models import Model, Sequential
from keras.layers import Dense, Dropout,Input, AveragePooling2D, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from keras.callbacks import ReduceLROnPlateau,ModelCheckpoint
from keras.optimizers import SGD
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import itertools
import seaborn as sns
import time
import os
import pandas as pd
import math
import keras
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.applications.densenet import DenseNet121
from keras.models import Model, Sequential
from keras.layers import Dense, Dropout,Input, AveragePooling2D, Activation,GlobalAveragePooling2D
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization, AvgPool2D
from keras.callbacks import ReduceLROnPlateau,ModelCheckpoint
from keras.optimizers import SGD
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import itertools
import seaborn as sns
%matplotlib inline
```

Kode Program 4.3 Instalisasi *library*

Pada pengimplementasian ini terdapat beberapa *library* seperti os, pandas, math, keras, numpy, tensorflow, matplotlib, time, seaborn, itertools, dan sklearn.

4.3.2 Fungsi *Split* Dataset

Dataset yang tersedia selanjutnya dilakukan pembagian dimana jenis dataset dibagi menjadi 3 yaitu *training*, *validation*, dan *testing*. Fungsi split dataset ditunjukkan pada Kode Program 4.4.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.10, random_state=300)


X_test, X_val, y_test, y_val = train_test_split(
    X_test, y_test, test_size=0.10, random_state=100)
```

Kode Program 4.4 Fungsi *split* dataset

Pada baris pertama adalah proses *split* dataset menjadi data *train* dan *test* sedangkan baris kedua adalah proses *split* dataset menjadi data test dan *validation* menggunakan fungsi `train_test_split` dengan parameter `test_size` yang berfungsi memisahkan data *testing* sebesar 10%, dan `random_state` yang berfungsi untuk mengontrol pengacakan yang diterapkan pada data sebelum menerapkan pemisahan.

4.3.3 Fungsi Inisialisasi *Input* CNN

Citra yang menjadi *inputan* diinisialisasi sesuai dengan ukuran pada saat proses *resize*. Fungsi inisialisasi *input* CNN ditunjukkan pada Kode Program 4.5.

```
dim = (256, 256)
channel = (3,)
input_shape = dim + channel
batch_size = 16
```

Kode Program 4.5 Fungsi inisialisasi *input* CNN

Inisialisasi input akan disesuaikan dengan ukuran citra yang telah melewati proses *resize*. Pada *variable* `dim` akan diatur sesuai dengan dimensi citra yaitu 256x256 dan *variable* `channel` akan diatur menjadi 3 yang dimaksud adalah 3 channel pada matriks citra yaitu RGB (*Red*, *Green*, *Blue*). Pada *variable* `batch_size` berfungsi menyebarkan jumlah sampel data ke jaringan dalam satu kali *epoch*.

4.3.4 Fungsi Memuat Data

Data citra yang terbagi menjadi 3 yaitu data *training*, *testing*, dan *validation* memiliki *directory* yang berbeda. Fungsi memuat data ditunjukkan pada Kode Program 4.6.

```
train_generator = train_datagen.flow_from_directory('/content/drive/MyDrive/file_gambar_paru/dataset_CNN/train/',
                                                    batch_size=batch_size,
                                                    class_mode='categorical',
                                                    shuffle=True)

val_generator = val_datagen.flow_from_directory('/content/drive/MyDrive/file_gambar_paru/dataset_CNN/val/',
                                                batch_size=batch_size,
                                                class_mode='categorical',
                                                shuffle=True)

test_generator = test_datagen.flow_from_directory('/content/drive/MyDrive/file_gambar_paru/dataset_CNN/test/',
                                                  batch_size=batch_size,
                                                  class_mode='categorical',
                                                  shuffle=True)

num_class = test_generator.num_classes
labels = train_generator.class_indices.keys()
```

Kode Program 4.6 Fungsi memuat data

Fungsi memuat data citra dari masing-masing *directory* menggunakan fungsi `flow_from_directory` pada *training*, *testing*, dan *validation* menggunakan parameter yang sama. Parameter pertama adalah *directory* yang mendefinisikan data citra berasal. Parameter kedua adalah batch_size berfungsi menyebarkan jumlah sampel data ke jaringan dalam satu kali *epoch*. Parameter ketiga adalah `class_mode` mendefinisikan jenis *class* yang dimiliki. Parameter keempat adalah `shuffle` bernilai *true* yang mendefinisikan data dalam *directory* tersebut diacak sehingga tidak sesuai urutan. Fungsi `num_classes` mendefinisikan jumlah *class* yang terdapat pada data *testing* dan disimpan pada *variable num_class*. Semua jumlah *class* di data *train* disimpan pada *variable labels*.

4.3.5 Fungsi Model CNN

Proses pemodelan dapat dilakukan setelah semua persiapan telah selesai. Fungsi model CNN ditunjukkan pada Kode Program 4.7.

```
model = Sequential()
#Convolutional
model.add(Conv2D(filters=128, kernel_size=(11, 11), strides=(2,2), input_shape=(256,256,3), activation='relu'))
model.add(BatchNormalization())

#Pooling
model.add(MaxPooling2D(pool_size=(3, 3),strides=(2, 2)))

#Convolutional
model.add(Conv2D(filters=128, kernel_size=(5, 5), padding='same', activation='relu'))

model.add(BatchNormalization())
#Pooling
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))

#Convolutional
model.add(Conv2D(filters=256, kernel_size=(3, 3), padding='same', activation='relu'))

#Convolutional
model.add(Conv2D(filters=384, kernel_size=(3, 3), padding='same', activation='relu'))

#Pooling
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))

#Convolutional
model.add(Conv2D(filters=256, kernel_size=(3, 3), padding='same', activation='relu'))

#Pooling
model.add(MaxPooling2D(pool_size=(3, 3),
                       strides=(2, 2)))

model.add(Flatten()) #Flatten

model.add(Dense(1024, activation='relu')) #Hidden Layer
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))#Hidden Layer
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(num_class, activation='softmax'))#Fully connected dan softmax
```

Kode Program 4.7 Fungsi model CNN

Pada model CNN, layer ekstraksi memiliki *convolution layer* dengan tambahan fungsi aktivasi ReLU, *padding same*, dan *stride*, *max pooling* dan diakhiri dengan *flatten layer*. Layer klasifikasi memiliki *hidden layer* yang memiliki 1024 dan 512 *node* menggunakan aktivasi ReLU. Penambahan *dropout* juga bertujuan mencegah terjadi *overfitting* dalam jaringan saraf tiruan, dan diakhiri dengan *output layer* dengan jumlah banyaknya kelas dan aktivasi *softmax*.

4.3.6 Fungsi *Learning Rate* dan *Optimizer*

Pada fungsi ini mendefinisikan *learning rate* dan *optimizer* yang digunakan pada saat proses *training*. Fungsi *learning rate* dan *optimizer* ditunjukkan pada Kode Program 4.8.

```
model.compile(loss='categorical_crossentropy',
              optimizer=SGD(learning_rate=1e-1, momentum=0.9, nesterov=True),
                metrics=['accuracy'])
```

Kode Program 4.8 Fungsi *learning rate* dan *optimizer*

Pada penelitian ini setelah pendefinisian model selanjutnya model di*compile* dengan beberapa fungsi yaitu `loss` menggunakan `categorial_crossentropy` karena data citra memiliki banyak kelas. Fungsi selanjutnya adalah `optimizer` menggunakan `Stochastic Gradient Descent` (SGD), dengan parameter `learning_rate`, `momentum`, `nesterov`, dan fungsi `metrics` digunakan untuk menentukan opsi metric yang ditampilkan, pada penelitian ini digunakan *accuracy*.

4.3.7 Fungsi Learning Rate Reduction

Pada fungsi ini bertujuan mengurangi kecepatan pembelajaran saat *metric* berhenti meningkat. Fungsi *learning rate reduction* ditunjukkan pada Kode Program 4.9.

```
filepath="/content/drive/MyDrive/file_gambar_paru/CNN.best.hdf5"
#callbacks
metrics = 'val_accuracy'
learning_rate_reduction = ReduceLROnPlateau(monitor=metrics,
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.0001)
checkpoint_save = ModelCheckpoint(filepath, monitor=metrics, verbose=1, save_best_only=True, mode='max')

callbacks_list = [checkpoint_save,learning_rate_reduction]
```

Kode Program 4.9 Fungsi *learning rate reduction*

Pada penelitian ini proses *learning rate reduction* menggunakan fungsi `ReduceROnPlateu` yang memiliki beberapa parameter yaitu `monitor`, `patience`, `verbose`, `factor`, dan `min_lr`. Parameter pertama yaitu monitor yang bertujuan untuk memantau kuantitas pada *metrics*, pada penelitian ini *metrics* yang digunakan yaitu *val_accuracy*. Parameter kedua yaitu `patience` yang merupakan jumlah *epochs* tanpa perbaikan setelah tingkat pemberalajaran berkurang. Parameter ketiga yaitu `verbose` yang bertujuan menampilkan progress bar dimana 1 karena kita dapat melihat proses dan hasil dari setiap *steps*. Parameter keempat yaitu `factor` yang dimana tingkat pembelajaran akan berkurang. Parameter terakhir yaitu `min_lr` yang dimana batas minimal pada *learning rate*. Fungsi `ReduceLROnPlateau` akan disimpan pada *variable learning rate reduction*. Fungsi selanjutnya yaitu `ModelCheckPoint` yang memungkinkan

kita untuk menentukan di mana harus memeriksa titik – titik bobot model. Pada fungsi `ModelCheckPoint` terdapat beberapa parameter yaitu `filepath`, `monitor`, `verbose`, `save_best_only`, dan `mode`. Pada parameter filepath merupakan tempat dimana model akan disimpan ke dalam directory. Parameter selanjutnya yaitu `save_best_only` yang bertujuan untuk menyimpan hasil yang terbaik pada *epoch* tertentu. Parameter selanjutnya yaitu mode bernilai *max* yang merupakan tingkat pembelajaran akan berkurang ketika kuantitas telah berhenti menurun. Fungsi `ModelCheckPoint` akan disimpan pada *variable checkpoint_save*. Setelah itu *variable learning rate reduction* dan *checkpoint_save* akan disimpan di dalam *variable callback_list*.

4.3.8 Fungsi *Training* dan *Testing* Model CNN

Pada proses ini dilakukan proses *training* dan *testing* pada model CNN. Fungsi *training* dan *testing* model CNN ditunjukkan pada Kode Program 4.10 dan Kode Program 4.11. Hasil dan visualisasi model CNN ditunjukkan pada Gambar 4.4 sampai 4.5.

```
epochs=10
num_folds = 5
#Define per-fold score containers
acc_per_fold_in_training = []
loss_per_fold_in_training = []

acc_per_fold_in_testing = []
loss_per_fold_in_testing = []
# Merge inputs and targets
x = np.concatenate((X_train, X_test), axis=0)
y = np.concatenate((y_train, y_test), axis=0)
# Define the K-fold Cross Validator

kfold = KFold(n_splits=num_folds, shuffle=True)
# K-fold Cross Validation model evaluation
fold_no = 1
for train, test in kfold.split(x, y):
```

Kode Program 4.10 Fungsi *K-Fold Cross Validation*

Pada penelitian proses training menggunakan *epochs* dimana berfungsi menentukan berapa kali model yang telah dibuat telah melihat dataset secara keseluruhan, pada proses ini *epochs* yang digunakan sebesar 100. Pada *K-Fold Cross Validation* menggunakan *variable num_folds* yang bertujuan sebagai *variable* untuk mengatur jumlah *fold* yang digunakan untuk proses *training*. Hasil

accuracy maupun *loss* pada proses *training* akan disimpan pada *variable* *acc_per_fold_in_training*, dan *loss_per_fold_in_training*. Selanjutnya hasil *accuracy* maupun *loss* pada proses *testing* akan disimpan pada *variable* *acc_per_fold_in_training*, dan *loss_per_fold_in_training*. Penggabungan dua atau lebih array sepanjang sumbu tertentu dalam proses ini menggunakan fungsi `np.concatenate`.

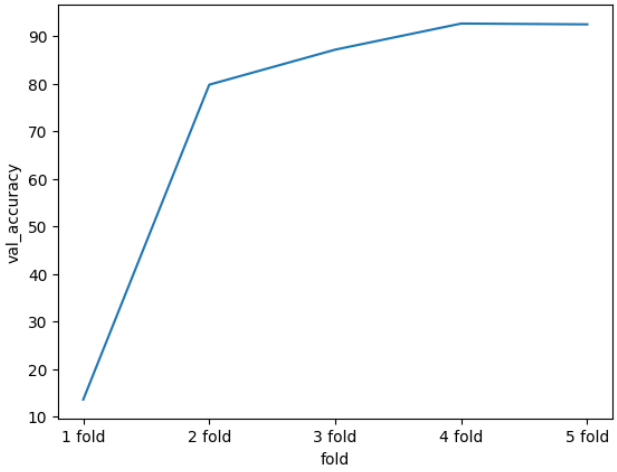
```
iterasi1 = model.fit(x=train_data,
                     steps_per_epoch = len(train_generator),
                     epochs=epochs,
                     validation_data=val_data,callbacks=callbacks_list,
                     validation_steps=len(val_generator),
                     shuffle=False,
                     verbose = 1)
Generate generalization metrics in training and testing
scores_training = model.evaluate(train_data,steps=len(train_generator),verbose=0)
print(f'Score in training for fold  {fold_no}: {model.metrics_names[0]} of
{scores_training[0]}; {model.metrics_names[1]} of {scores_training[1]*100}%')
acc_per_fold_in_training.append(scores_training[1] * 100)
loss_per_fold_in_training.append(scores_training[0])

scores_testing = model.evaluate(test_data,steps=len(test_generator),verbose=0)
print(f'Score in testing for fold  {fold_no}: {model.metrics_names[0]} of
{scores_testing[0]}; {model.metrics_names[1]} of {scores_testing[1]*100}%')
acc_per_fold_in_testing.append(scores_testing[1] * 100)
loss_per_fold_in_testing.append(scores_testing[0])
```

Kode Program 4.11 Fungsi Training dan Testing Model CNN

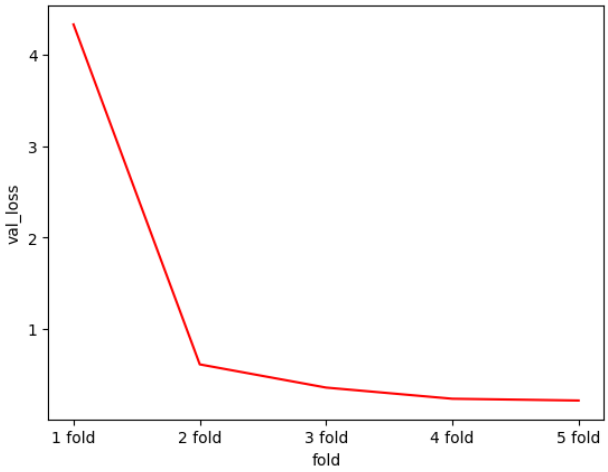
Proses selanjutnya yaitu menjalakan training dengan fungsi `model.fit` dengan beberapa parameter. Parameter pertama yaitu `train_data` yang diisi data *training*. Parameter kedua yaitu `step_per_epoch` untuk menentukan jumlah *steps* yang akan dilakukan tiap *epoch* untuk data *training* diisi dengan fungsi *len* untuk mengembalikan panjang jumlah data pada `train_generator`. Parameter ketiga yaitu `epoch` diisi dengan jumlah *epochs*. Parameter keempat yaitu `validation_data` diisi dengan data *validation*. Parameter kelima yaitu `callbacks` diisi dengan data yang ada di *variable* pada proses sebelumnya yaitu *callback_list*. Parameter keenam yaitu `validation_steps` untuk menentukan jumlah *steps* yang akan dilakukan tiap *epoch* untuk data *validation* diisi dengan fungsi *len* untuk mengembalikan panjang jumlah data pada `val_generator`. Parameter ketujuh yaitu `shuffle` bernilai *False* yang artinya data di dalam

directory tersebut tidak diacak sehingga sesuai urutan. Parameter kedelapan `verbose` yang bertujuan menampilkan progress bar dimana 1 karena kita dapat melihat proses dan hasil dari setiap *steps*. Setelah proses *training* berjalan selanjutnya yaitu proses *testing*. Pada proses menggunakan fungsi `model.evaluate` dengan tujuan melakukan perhitungan nilai *accuracy* dan nilai *loss*. Nilai *accuracy* dan *loss* pada data *training* akan ditampung pada *variable* `scores_training` yang memiliki beberapa parameter. Parameter pertama yaitu `train_data` yang diisi data *training*. Parameter kedua yaitu `steps` yang diisi untuk menentukan jumlah *steps* yang akan mengevaluasi data *training* yang diisi dengan fungsi *len* untuk mengembalikan panjang jumlah data pada `train_generator`. Selanjutnya hasil perhitungan nilai *accuracy* dan nilai *loss* akan dicetak. Selanjutnya nilai accuracy dan loss pada data *testing* akan ditampung pada *variable* `scores_testing` yang memiliki beberapa parameter. Parameter pertama yaitu `test_data` yang diisi data *testing*. Parameter kedua yaitu `steps` yang diisi untuk menentukan jumlah *steps* yang akan mengevaluasi data *testing* yang diisi dengan fungsi *len* untuk mengembalikan panjang jumlah data pada `test_generator`. Selanjutnya hasil perhitungan nilai *accuracy* dan nilai *loss* akan dicetak. Selanjutnya hasil waktu proses *training* akan dicetak dengan memanggil *variable start* yang sebelumnya sudah didefinisikan.

Pada Gambar 4.4 hasil *training dan testing* model CNN memperoleh akurasi pada saat *training* diperoleh 0.93 dan pada saat *testing* diperoleh 0.92 dengan konfigurasi 5 *Fold Cross Validation*. Hasil dan visualisasi model CNN ditunjukkan pada Gambar 4.4 sampai 4.5.



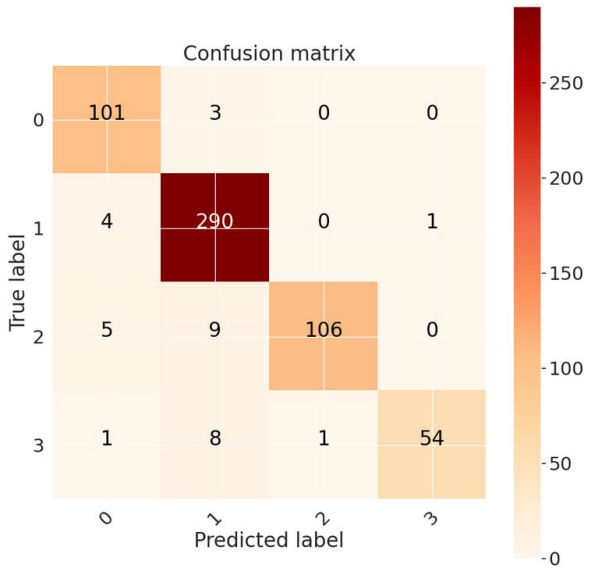
Gambar 4.4 Hasil *accuracy* pengujian model CNN



Gambar 4.5 Hasil *loss* pengujian model CNN

4.3.9 Evaluasi model CNN

Skenario terakhir ada evaluasi *accuracy*, *precision*, *recall* dan *f1-score* terhadap data testing dengan *confusion matrix*, yang bertujuan untuk mengetahui seberapa baik model dalam mengklasifikasikan data. Confusion matrix model CNN dan CNN tuning ditunjukkan pada Gambar 4.6 dan Gambar 4.7.



Gambar 4.6 Confusion matrix model CNN

Hasil *confusion matrix* pada Gambar 4.6 maka dapat dilakukan perhitungan evaluasi *accuracy*, *precision*, *recall*, dan f1-*score*. Pada kelas 0 (Bacterial Pneuomonia) model CNN berhahsil mengklasifikasikan data dengan benar berjumlah 101 data, kelas 1 (COVID-19) model CNN berhasil mengklasifikasikan dengan benar berjumlah 290 data, kelas 2 (Normal) model CNN berhasil mengklasifikasikan data dengan benar berjumlah 106 data, dan kelas 3 (Tuberculosis) model CNN berhasil mengklasifikasikan data dengan benar berjumlah 54 data. Hasil evaluasi model CNN ditunjukkan pada Gambar 4.7.

	precision	recall	f1-score
BacterialPneumonia	0.91	0.97	0.94
COVID-19	0.94	0.98	0.96
Normal	0.99	0.88	0.93
Tuberculosis	0.98	0.84	0.91
accuracy			0.95
macro avg	0.95	0.92	0.93
weighted avg	0.95	0.95	0.94

Gambar 4.7 Hasil evaluasi model CNN

Pada hasil evaluasi model CNN didapatkan *accuracy* 0.95, *precision* 0.95, *recall* 0.95, dan f1-*score* 0.94. Hasil di atas menunjukkan bahwa *accuracy* pada saat

testing dan evaluasi tidak beda jauh yang berarti model CNN yang dibangun berhasil.

4.4 Support Vector Machine

Data yang telah selesai dilakukan preprocessing selanjutnya akan diimplementasikan dengan metode SVM untuk mengetahui apakah metode SVM cocok digunakan pada klasifikasi penyakit paru atau tidak.

4.4.1 Instalisasi *Library*

Pada tahap ini beberapa *library* akan digunakan agar dapat melaksanakan proses modeling dengan lebih mudah. *Library* bantuan yang digunakan pada pengimplementasian ini ditunjukkan pada Kode Program 4.12.

```
import pylab as pl
import numpy as np
import os
import cv2
from matplotlib import pyplot as plt
import matplotlib.image as mplib
from tensorflow.keras import utils
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA as RandomizedPCA
from sklearn.svm import SVC
from time import time
```

Kode Program 4.12 Instalisasi library

Pada pengimplementasian ini terdapat beberapa *library* seperti os, pandas, numpy, cv2 matplotlib, time, keras, dan sklearn.

4.4.2 Fungsi Split Dataset

Dataset yang tersedia selanjutnya dilakukan pembagian dimana jenis dataset dibagi menjadi 2 yaitu *training,* dan *testing*. Fungsi split dataset ditunjukkan pada Kode Program 4.13.

```
Y = utils.to_categorical(labels, num_classes)
y = labels
X_train, X_test, y_train, y_test = train_test_split(img_data, y,
                                                    test_size=0.35,
                                                    shuffle = True, random_state=60)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Kode Program 4.13 Fungsi *Split Dataset*

Pada *variable* Y menggunakan fungsi utils.to_categorical karena data yang dimiliki memiliki banyak kelas. Selanjutnya *labels* data disimpan pada *variable* y. Proses *split* dataset yaitu memisahkan data *training* dan *testing* menggunakan fungsi train_test_split dengan parameter test_size yang berfungsi memisahkan data *testing* sebesar 20%, dan random_state yang berfungsi untuk mengontrol pengacakan yang diterapkan pada data sebelum menerapkan pemisahan.

4.4.3 Fungsi Principal Component Analysis

Dataset yang telah dilakukan *split* selanjutnya akan diekstraksi fitur menggunakan algoritma *principal component analysis*. Fungsi PCA ditunjukkan pada Kode Program 4.14.

```
n_components = 153
pca = RandomizedPCA(n_components=n_components, whiten=False).fit(X_train)
sum(pca.explained_variance_ratio_)
```

Kode Program 4.14 Fungsi PCA

Fungsi PCA yang diimplementasikan memiliki beberapa parameter. Pada baris pertama menggunakan parameter `n_components` yang artinya adalah jumlah maksimum komponen yang harus disimpan. Parameter kedua yaitu whiten yang bernilai *default* yaitu *false*, ketika bernilai *true* maka komponen vector dibagai dengan nilai tunggal untuk memastikan *output* yang tidak berkolerasi dengan varian unit komponen.

4.4.4 Fungsi Model SVM

Proses pemodelan dapat dilakukan setelah semua persiapan telah selesai. Fungsi model SVM ditunjukkan pada Kode Program 4.15 dan hasil fungsi model SVM ditunjukkan pada Gambar 4.8.

```
print("Fitting the classifier to the training set")
t0 = time()
param_grid = {
    'C': [1,5,10,15]
}
clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced',gamma='scale'),
                   param_grid,n_jobs=None, refit=True)
clf = clf.fit(X_train_pca, y_train)
print("done in %0.3fs" % (time() - t0))
print("Best estimator found by grid search:")
print(clf.best_estimator_)
```

Kode Program 4.15 Fungsi model SVM

Pada baris pertama *variable param_grid* memiliki dua parameter. Parameter pertama adalah `C` yang memiliki nilai 1, 5, 10, 15 dan parameter kedua adalah gamma yang memiliki nilai *scale*. Pada fungsi `GridSearchCV` diatur `kernel` dengan nilai rbf, `class_weight` dengan nilai balanced, `n_jobs` dengan nilai *None*, `refit` dengan nilai *true*, dan *variable param_grid* dimasukkan ke dalam fungsi ini.

```
Fitting the classifier to the training set
done in 1.567s
Best estimator found by grid search:
SVC(C=5, class_weight='balanced')
```

Gambar 4.8 Hasil fungsi model SVM

Gambar di atas merupakan hasil saat fungsi model SVM dijalankan, fungsi `GridSearchCV` akan memilih nilai dari parameter `C` dan `class_weight` yang dianggap optimal.

4.4.5 Fungsi Testing Model SVM

Pada proses ini dilakukan proses *testing* pada model SVM. Fungsi *testing* model SVM ditunjukkan pada Kode Program 4.16.

```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_pred)
print(score)
```

Kode Program 4.16 Fungsi *testing* model SVM

Pada proses *testing* menggunakan *library* bantuan sklearn. Fungsi *accuracy_score* digunakan untuk mengetahui akurasi saat proses *testing* yang didapat dari parameter `y_test` dan `y_pred`. Hasil *testing* model SVM ditunjukkan pada Gambar 4.9.

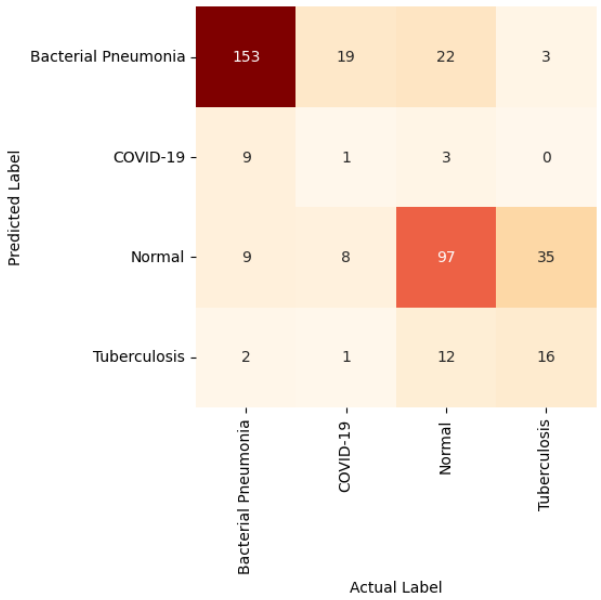
```
0.6846153846153846
```

Gambar 4.9 Hasil *testing* model SVM

Hasil *testing* model SVM yang dilakukan pada saat pengujian mendapatkan akurasi 0.68, tentunya hasil tersebut belum lebih baik dari hasil model CNN.

4.4.6 Fungsi Evaluasi Model SVM

Skenario terakhir ada evaluasi *accuracy, precision, recall* dan f1-*score* terhadap data *testing* dengan *confusion matrix*, yang bertujuan untuk mengetahui seberapa baik model dalam mengklasifikasikan data. *Confusion matrix* model SVM ditunjukkan pada Gambar 4.10.



Gambar 4.10 *Confusion matrix* model SVM

Hasil *confusion matrix* pada Gambar 4.10 maka dapat dilakukan perhitungan evaluasi *accuracy, precision, recall*, dan f1-*score*. Pada kelas (Bacterial Pneuomonia) model SVM berhahsil mengklasifikasikan data dengan

benar berjumlah 153 data, kelas (COVID-19) model SVM berhasil mengklasifikasikan dengan benar berjumlah 1 data, kelas (Normal) model SVM berhasil mengklasifikasikan data dengan benar berjumlah 97 data, dan kelas (Tuberculosis) model SVM berhasil mengklasifikasikan data dengan benar berjumlah 16 data. Hasil evaluasi model SVM ditunjukkan pada Gambar 4.11.

	precision	recall	f1-score
Bacterial Pneumonia	0.78	0.88	0.83
COVID-19	0.08	0.03	0.05
Normal	0.65	0.72	0.69
Tuberculosis	0.52	0.30	0.38
accuracy			0.68
macro avg	0.51	0.48	0.48
weighted avg	0.65	0.68	0.66

Gambar 4.11 Hasil evaluasi model SVM

Pada hasil evaluasi model SVM didapatkan *accuracy* 0.68, *precision* 0.65, *recall* 0.68, dan f1-*score* 0.66. Hasil di atas menunjukkan bahwa *accuracy* pada saat *testing* dan evaluasi tidak beda jauh yang berarti model SVM yang dibangun berhasil.

4.5 Benchmark Model CNN dan SVM

Pada model CNN dan SVM dilakukan *benchmark* dari hasil evaluasi dengan tujuan apakah model CNN atau SVM yang cocok digunakan untuk proses pengenalan ekspresi wajah. Benchmark evaluasi model CNN dan SVM ditunjukkan pada Tabel 4.2.

Tabel 4.2 Benchmark model CNN dan SVM

Model	*Accuracy*	*Precision*	*Recall*	F1-*score*
CNN	0.95	0.95	0.95	0.94
SVM	0.68	0.65	0.68	0.66

Dari hasil *benchmark* yang ditunjukkan pada Tabel 4.2 model CNN yang dibangun berhasil mengungguli model SVM. Dari hasil eksperimen yang telah dilakukan berikut beberapa alasan yang mungkin bisa sebagai pendukung mengapa model CNN yang dibangun dapa mengunguli model SVM.

1. Ekstraksi Fitur Otomatis: CNN secara otomatis mengekstraksi fitur-fitur yang relevan dari data gambar melalui proses pembelajaran, sedangkan SVM memerlukan fitur-fitur yang sudah di-ekstrak dengan hati-hati sebelumnya. Proses ekstraksi fitur secara otomatis oleh CNN dapat mengatasi masalah kompleksitas dan kerumitan dari data gambar yang sangat besar.
2. Kemampuan Spasial: CNN mempertahankan informasi spasial dari gambar melalui penggunaan layer konvolusi dan pooling. Hal ini memungkinkan CNN untuk memahami pola-pola spasial dalam gambar dengan lebih baik daripada SVM yang hanya memperlakukan data gambar sebagai vektor.
3. Invariansi Terhadap Pergeseran dan Rotasi: CNN secara alami cenderung lebih invarian terhadap pergeseran dan rotasi gambar karena konsep layer konvolusi. Hal ini membuat CNN lebih cocok untuk tugas-tugas seperti pengenalan objek dalam gambar yang tidak selalu memiliki orientasi atau posisi yang tetap.
4. Kemampuan Hierarkis: CNN memiliki struktur hierarkis yang memungkinkan mereka untuk mempelajari representasi yang semakin kompleks dari data gambar. Ini berarti CNN bisa memahami konsep-konsep yang lebih kompleks dan abstrak dalam gambar.
5. Efisiensi Penggunaan Data: CNN dapat memanfaatkan struktur data spasial gambar untuk mengurangi jumlah parameter yang diperlukan untuk dilatih, sehingga lebih efisien dalam penggunaan data.

Meskipun demikian, SVM tetap memiliki kegunaannya dalam beberapa kasus, terutama jika data yang digunakan sudah memiliki fitur-fitur yang terdefinisi dengan baik dan jumlah sampel yang relatif kecil.