BIBLIOGRAPHY

Andriani, A. (2013). *Sistem Pendukung Keputusan Berbasis Decision Tree Dalam Pemberian Beasiswa Studi Kasus: Amik 'Bsi Yogyakarta,'*. Nasional Teknologi Informasi Dan Komunikasi, 163-168.

Angki, A., & Subhan, A. (2011). *Rancang Bangun Rtp Chunk-Packet Encapsulator Data Av Stream Format Rtp Pada Multi-Source Streaming Server*. EEPIS Final Project.

Bovik, A. (2005). Handbook of Image and Video Processing. *A volume in Communications, Networking and Multimedia*. Department Of Electrical and Computer Engineering, The University of Texas at Austin, Austin, Texas

Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.".

Forson, E. (2017). *Understanding SSD MultiBox — Real-Time Object Detection in Deep Learning*. Towards Data Science. Retrieved from https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab

Gorunescu, F. (2011). Intelligent Systems Refrence Library. *Data Mining: Concepts, Models and Techniques*. (Vol. 12): Springer Science & Business Media.

Han, J., Kamber, M., & Pei, J. (2011). The Morgan Kaufmann Series in Data Management Systems. *Data Mining Concepts and Techniques Third Edition*. (Vol. 5): Morgan Kaufmann.

Hayuningtyas, R. Y. (2017). *Aplikasi Filtering of Spam Email Menggunakan Naïve Bayes*. IJCIT (Indonesian Journal on Computer and Information Technology, 2(1).

Herlambang, M. (2019). *Deep Learning: Convolutional Neural Networks (app)*. Retrieved from https://www.megabagus.id/deep-learning-convolutional-neural-networks-aplikasi/

Hidayatullah, P. (2017). *Pengolahan CItra Digital Teori dan Aplikasi Nyata*. Bandung: Informatika.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., . . . Adam, H. (2017). *Mobilenets: Efficient convolutional neural networks for mobile vision applications.*

Huang, C., Wang, Y., Li, X., Ren, L., Zhao, J., Hu, Y., . . . Gu, X. (2020). *Clinical Features of Patients Infected with 2019 Novel Coronavirus in Wuhan, China*. The Lancet, 395(10223), 497-506.

Jalled, F., & Voronkov, I. (2016). *Object Detection Using Image Processing*. Moscow Institute of Physics & Technology. Department of Radio Engineering & Cybernetics.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *Imagenet Classification with Deep Convolutional Neural Networks*. Communications of the ACM, 25, 1097-1105.

Liu, K.-C., Xu, P., Lv, W.-F., Qiu, X.-H., Yao, J.-L., Gu, J.-F., & Wei, W. (2020). *CT Manifestations of Coronavirus Disease-2019: a Retrospective Analysis of 73 Cases by Disease Severity*. European journal of radiology, 126, 108941.

Lu, Y. (2017). *Deep Neural Networks and Fraud Detection.* Mathematics. Uppsala University. Sweden.

Mahmud, K. H., Adiwijaya, A., & Al Faraby, S. (2019). *Klasifikasi Citra Multi-kelas Menggunakan Convolutional Neural Network*. eProceedings of Engineering, 6(1).

Manning, C. E. (1996). *Digital Video Compression*. Retrieved from http://www.newmediarepublic.com/dvideo/compression/adv02.html

Marino, K. (2020). *Early Face Mask Policies Curbed COVID-19's Sread, Acording to 198-Cuntry Aalysis*. Children's Hospital of Richmond at VCU, VCU NEWS. Retrieved from https://news.vcu.edu/article/Early_face_mask_policies_curbed_COVID19s_spread_according_to

Maulana, A. (2020). *Efektivitas Pembatasan Sosial Berskala Besar (PSBB) Sebagai Salah Satu Upaya Menekan Penyebaran Covid-19 Di Dki Jakarta.* Ilmu Sosial. Universitas Negri Jakarta. Jakarta.

McCulloch, W. S., & Pitts, W. (1943). *A Logical Calculus of The Ideas Immanent in Nervous Activity*. The bulletin of mathematical biophysics, 5(4), 115-133.

Nalwan, Agustinus. (1997). *Pengolahan Gambar Secara Digital*. Jakarta: Elex Media Komputindo.

Perkovic, L. (2012). An Application Development Focus. *Introduction to Computing Using Python*. DePaul University. Illinois: Wiley.

Powers, D. M. (2020). *From Precision, Recall and F-Measure To ROC, Informedness, Markedness & Correlation*. International Journal of Machine Learning Technology.

Rossi, A. (2016). *Analisis Dan Implementasi Algoritma 3D-DCT Dengan Menggunakan Perkodean Cabac Pada Kompresi Video H. 265/HEVC* Doctoral dissertation, Universitas Komputer Indonesia

Samuel, A. L. (1959). *Some Studies in Machine Learning Using the Game of Checkers*. IBM Journal of research development, 3(3), 210-229. doi:10.1016/0066-4138(69)90004-4

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition, 4510-4520.

Sanger, J. W. L. (2021). *Face Masks During the COVID-19 Pandemic*. Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Face_masks_during_the_COVID-19_pandemic#cite_ref-2

Sanjaya, J., & Ayub, M. (2020). *Augmentasi Data Pengenalan Citra Mobil Menggunakan Pendekatan Random Crop, Rotate, dan Mixup*. Jurnal Teknik Informatika dan Sistem Informasi, 6(2).

Sena, S. (2017). *Pengenalan Deep Learning Part 7: Convolutional Neural Network (CNN)*. Deep Reinforcement Learning Student. Medium. Retrieved from https://medium.com/@samuelsena/pengenalan-deep-learning-part-7-convolutional-neural-network-cnn-b003b477dc94

Sokolova, M., & Lapalme, G. (2009). *A Systematic Analysis of Performance Measures for Classification Tasks*. Information processing management, 45(4), 427-437.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. The journal of machine learning research, 15(1), 1929-1958.

Sutoyo, T d. (2009). *Teori Pengolahan Citra Digital*. Yogyakarta: Andi.

Suyanto, K. N. R., & Mandala, S. (2019). *Deep Learning Modernisasi Machine Learning Untuk Big Data*. Bandung: Informatika.

Taufiq, I. (2018). *Deep Learning for Detection Motor Vehicle Number Signs Using Convolutional Neural Network Algorithm with Python and Tensorflow.* Sistem Informasi. STMIK AKAKOM.

Vargas, R., & Lourdes, R. (2017). *Deep Learning: Previous and Present Applications*. Journal of Awareness, 2(Special 3), 11-20.

Vedaldi, A., & Lenc, K. (2015). *Matconvnet: Convolutional neural networks for matlab*. Proceedings of the 23rd ACM international conference on Multimedia, 689-692.

Verma, S., Dhanak, M., & Frankenfield, J. (2020). *Visualizing The Effectiveness of Face Masks in Obstructing Respiratory Jets*. Physics of Fluids, 32(6), 061708.

Visalini, S. (2017). *Traffic Sign Recognition Using Convolutional Neural Network*. International Jurnal of Innovative Research in Computer Communication Engineering, 5.

Wicaksono, A. Y., Suciati, N., Fatichah, C., Uchimura, K., & Koutaki, G. (2017). *Modified Convolutional Neural Network Architecture for Batik Motif Image Classification*. IPTEK Journal of Science, 2(2).

Woodruff, J., Santhanam, L., & Thoet, A. (2020). *What Dr. Fauci Wants You to Know About Face Masks and Staying Home as Airus Spreads*. PBS NewsHour. Retrieved from https://www.pbs.org/newshour/show/what-dr-fauci-wants-you-to-know-about-face-masks-and-staying-home-as-virus-spreads

Woods, A. (2020). *Britain Faces an Anxiety Crisis as People Return to Work*. Bdaily News. Retrieved from https://bdaily.co.uk/articles/2020/06/22/britain-faces-an-anxiety-crisis-as-people-return-to-work

Zhi, T., Duan, L.-Y., Wang, Y., & Huang, T. (2016). *Two-Stage Pooling of Deep Convolutional Features for Image retrieval*. 2016 IEEE International Conference on Image Processing (ICIP), 2465-2469.

Zufar, M. (2016). *Convolutional Neural Networks Untuk Pengenalan Wajah Secara Real-time.* Matematika. Institut Technology Sepuluh Nopember. Surabaya.

ATTACHMENT

Source Code

Creating the model and model evaluation

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import
preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
import itertools
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import pandas as pd
import sklearn
import matplotlib.image as mpimg
import random
import os

INIT_LR = 1e-4
EPOCHS = 20
BS = 32

print("[INFO] loading images...")
```

```
imagePaths =
list(paths.list_images('D:\Documents\IMPORTANT\SKRIPSI\PROJECT
SKRIPSI\FACE MASK DETECTION MASTER\Face-Mask-Detection-master-
FIX\dataset'))
data = []
labels = []

for imagePath in imagePaths:

    label = imagePath.split(os.path.sep)[-2]

    image = load_img(imagePath, target_size = (224, 224))
    image = img_to_array(image)
    image = preprocess_input(image)

    data.append(image)
    labels.append(label)

data = np.array(data, dtype = "float32")
labels = np.array(labels)

lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)

(trainX, testX, trainY, testY) = train_test_split(data, labels,
    test_size = 0.20, stratify = labels, random_state = 42)

aug = ImageDataGenerator(
    rotation_range = 20,
    zoom_range = 0.15,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.15,
    horizontal_flip = True,
    fill_mode = "nearest")

baseModel = MobileNetV2(weights = "imagenet", include_top = False,
    input_tensor = Input(shape = (224, 224, 3)))

headModel = baseModel.output
headModel = AveragePooling2D(pool_size = (7, 7))(headModel)
headModel = Flatten(name = "flatten")(headModel)
headModel = Dense(128, activation = "relu")(headModel)
headModel = Dropout(0.5)(headModel)
```

```
headModel = Dense(2, activation = "softmax")(headModel)

early_stopping = EarlyStopping(monitor = 'val_accuracy', patience=15)
checkpoint = ModelCheckpoint('checkpoint/check', monitor =
'val_accuracy', verbose = 1, save_best_only = True, save_weights_only
= True, save_freq = 'epoch')

model = Model(inputs = baseModel.input, outputs = headModel)

for layer in baseModel.layers:
    layer.trainable = False

print("[INFO] compiling model...")
opt = Adam(lr = INIT_LR, decay = INIT_LR / EPOCHS)
model.compile(loss = "binary_crossentropy", optimizer = opt,
    metrics = ["accuracy"])

print("[INFO] training head...")
H = model.fit(
    aug.flow(trainX, trainY, batch_size = BS),
    steps_per_epoch = len(trainX) // BS,
    validation_data = (testX, testY),
    validation_steps = len(testX) // BS,
    epochs = EPOCHS,
    callbacks = [early_stopping, checkpoint])

print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size = BS)

predIdxs = np.argmax(predIdxs, axis = 1)

print(classification_report(testY.argmax(axis = 1), predIdxs,
    target_names = lb.classes_))

N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label = "train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label = "val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label = "train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label =
"val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
```

```
plt.legend(loc = "lower left")

model.summary()

epoch_result = pd.DataFrame([H.history['accuracy'],
H.history['val_accuracy'],
                             H.history['loss'], H.history['val_loss'],
],
                            columns = np.arange(1,EPOCHS + 1,step =
1),
                            index = ['accuracy', 'val_accuracy',
'loss', 'val_loss']).T
epoch_result.index.name = 'epoch'
for col in epoch_result:
    if col == 'loss' or col == 'val_loss':
        continue
    epoch_result[col] = epoch_result[col] * 100
epoch_result = epoch_result.round(2)
epoch_result

plt.figure(figsize = (12,8))
plt.plot(epoch_result['accuracy'],'-o', label = 'training', color =
'#ba0f28')
plt.plot(epoch_result['val_accuracy'], '-o', label = 'validation',
color = '#1f1fde')
plt.title('MobileNetV2 accuracy')
plt.ylim(80,100.5)
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.xticks(np.arange(1,EPOCHS + 1,step = 1))
plt.grid()
plt.legend()

plt.figure(figsize = (12,8))
plt.plot(epoch_result['loss'], '-o', label = 'training', color =
'#ba0f28')
plt.plot(epoch_result['val_loss'], '-o', label = 'validation', color =
'#1f1fde')
plt.title('MobileNetV2 loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.xticks(np.arange(1, EPOCHS + 1, step = 1))
plt.grid()
plt.legend()
```

```
def plot_confusion_matri(cm, classes,normalize = False,title =
'Confusion matrix',cmap = plt.cm.Reds):
    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis = 1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment = "center",
            color = "white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

labels = testY.argmax(axis = 1)
testp = model.predict(testX, batch_size = BS)
print(testp.argmax(axis = 1),end = " ")
cm = confusion_matrix(labels,testp.argmax(axis = 1))
cmlabels = ['with mask','without mask']
plot_confusion_matri(cm,cmlabels)

labels = trainY.argmax(axis = 1)
testp = model.predict(trainX, batch_size = BS)
print(testp.argmax(axis = 1),end = " ")
cm = confusion_matrix(labels,testp.argmax(axis = 1))
cmlabels = ['with mask','without mask']
plot_confusion_matri(cm,cmlabels)
```

System Face mask detection

```
from tensorflow.keras.applications.mobilenet_v2 import
preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
from playsound import playsound
import numpy as np
import argparse
import imutils
import time
import cv2
import os

def detect_and_predict_mask(frame, faceNet, maskNet):
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300),
        (104.0, 177.0, 123.0))

    faceNet.setInput(blob)
    detections = faceNet.forward()

    faces = []
    locs = []
    preds = []

    for i in range(0, detections.shape[2]):
        confidence = detections[0, 0, i, 2]

        if confidence > args["confidence"]:
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            (startX, startY) = (max(0, startX), max(0, startY))
            (endX, endY) = (min(w - 1, endX), min(h - 1, endY))

            face = frame[startY:endY, startX:endX]
            face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
            face = cv2.resize(face, (224, 224))
            face = img_to_array(face)
            face = preprocess_input(face)

            faces.append(face)
            locs.append((startX, startY, endX, endY))
```

```
    if len(faces) > 0:
        faces = np.array(faces, dtype="float32")
        preds = maskNet.predict(faces, batch_size=32)

    return (locs, preds)

ap = argparse.ArgumentParser()
ap.add_argument("-f", "--face", type=str,
    default="face_detector",
    help="path to face detector model directory")
ap.add_argument("-m", "--model", type=str,
    default="mask_detector.model",
    help="path to trained face mask detector model")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
    help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

print("[INFO] loading face detector model...")
prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
weightsPath = os.path.sep.join([args["face"],
    "res10_300x300_ssd_iter_140000.caffemodel"])
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

print("[INFO] loading face mask detector model...")
maskNet = load_model(args["model"])

print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)

noMask = 0
yesMask = 0
frameId = 0
frame_dict = dict()

while True:
    frame = vs.read()
    frame = imutils.resize(frame, width=400)

    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)

    for (box, pred) in zip(locs, preds):
        (startX, startY, endX, endY) = box
        (mask, withoutMask) = pred
```

```
        label = "Mask" if mask > withoutMask else "No Mask"
        color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

        if mask > withoutMask:
            yesMask += 1
        else:
            noMask += 1

        label = "{}: {:.2f}%".format(label, max(mask, withoutMask) *
100)

        cv2.putText(frame, label, (startX, startY - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
        cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

    frame_dict[frameId] = (yesMask,noMask)
    yesMask = 0
    noMask = 0
    textMask = "People wearing a Mask: " + str(frame_dict[frameId][0])
    textNoMask = "People not wearing a Mask: " +
str(frame_dict[frameId][1])
    locationYesMask = (10,25)
    locationNoMask = (10,45)
    cv2.putText(frame, textMask, locationYesMask,
                cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 255, 0), 2)
    cv2.putText(frame, textNoMask, locationNoMask,
                cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
    frameId += 1
    if frameId > 5:
        frameId = 0

    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    if key == ord("q"):
        break

cv2.destroyAllWindows()
vs.stop()
```