# IMPLEMENTASI CONVOLUTIONAL NEURAL NETWORK (CNN) UNTUK ILLUMINATION-INVARIANT FACE RECOGNITION MENGGUNAKAN DATASET EXTENDED YALE FACE DATABASE B

Jonathan Imago Dei Gloriawan

*Program Studi Teknik Komputer, Fakultas Teknik, Universitas Diponegoro*
*Jalan Prof. Sudharto, Tembalang, Semarang, Indonesia 50275*

***Abstrak** – Pengenalan wajah akan sangat berperan dalam mendorong produktivitas operasional dan menjaga keselamatan publik. Namun, untuk mewujudkannya pengenalan wajah harus benar-benar berfungsi dengan baik dan efektif. Dalam hal ini, pengenalan wajah harus mampu mengenali wajah seseorang dalam berbagai pose dan kondisi pencahayaan. Penelitian ini melakukan pembangunan pengenalan wajah untuk pose dan kondisi pencahayaan yang beragam. Dataset yang digunakan memanfaatkan data dari Extended Yale Face Database B, yang kami ambil 5 sub-database-nya dengan jumlah total sebanyak 2925 gambar. Lalu, digunakan 5 arsitektur jaringan saraf yang nantinya akan dibandingkan, yaitu FaceNet, Pose Invariant, ResNet Dorian Lazar, ResNet50, dan InceptionV3. Kelima arsitektur akan diuji dengan 2 parameter utama, yaitu pembagian data train dan test serta variasi kernel SVM. Di penelitian ini, kami juga menggunakan metode face embedding dan memanfaatkan algoritma Support Vector Machine dalam mengolah data. Hasil yang diperoleh dari penelitian ini adalah arsitektur FaceNet dan ResNet50 merupakan arsitektur jaringan saraf terbaik untuk pengenalan wajah dengan variasi pose dan kondisi pencahayaan. FaceNet dengan konfigurasi split data 90% data train dan 10% data test dengan kernel polynomial memiliki akurasi 99.24% untuk train dan 98.29% untuk test. ResNet50 dengan konfigurasi split data 90% data train dan 10% data test dengan kernel polynomial memiliki akurasi terbaik, yaitu 98.4% untuk train dan 97.95% untuk test.***

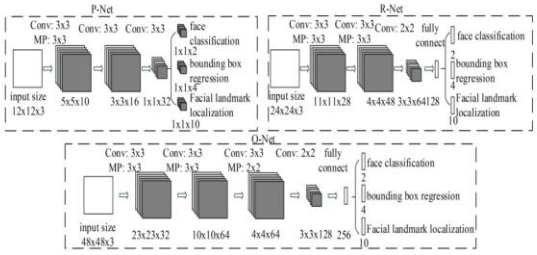
***Kata Kunci** – Face Recognition, FaceNet, ResNet50, InceptionV3, Face Embedding, SVM.*

## I. PENDAHULUAN

*Face recognition* adalah salah satu jenis sistem identifikasi "biometrik" yang mengidentifikasi wajah sesorang. Sistem ini mengidentifikasi seseorang dengan fitur-fitur khusus pada tubuh maupun DNA yang membedakan satu orang dengan orang lainnya. *Face recognition* akan memiliki peranan penting ke depannya dalam rangka untuk mendorong produktivitas operasional dan menjaga keselamatan publik. Face recognition akan lebih mudah dibangun dan diterapkan ketika wajah seseorang dalam kondisi yang prima, dalam artian pose wajah menghadap kamera, pencahayaan cukup, serta tidak ada objek lain yang menggangu baik latar belakang maupun objek-objek kecil lainnya. Namun, akan sangat tidak efektif apabila pengenalan wajah hanya mampu digunakan untuk gambar wajah yang prima, pada kenyataanya wajah sesorang akan tertangkap oleh kamera dalam berbagai pose dan kondisi pencahayaan, serta terdapat objek lain selain wajah.

Penelitian ini bertujuan untuk mampu melakukan pengenalan wajah pada pose dan kondisi pencahayaan yang bervariasi. Penelitian ini dilakukan dengan menggunakan 5 arsitektur jaringan saraf, yaitu FaceNet, ResNet50, InceptionV3, Pose Invariant Model, dan ResNet Dorian Lazar. FaceNet, ResNet50, dan InceptionV3 adalah *pre-trained* model, sedangkan Pose Invariant Model dan ResNet Dorian Lazar merupakan model sekuensial yang kami bangun sendiri namun tetap mengacu pada sumber atau artikel masing-masing model.

A. MTCNN

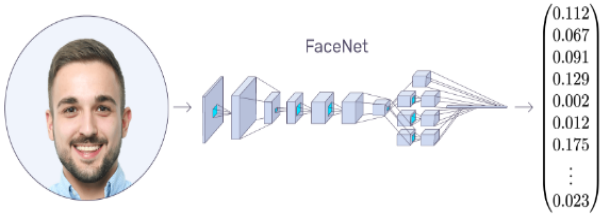



**Gambar 1** MTCNN

MTCNN atau *Multi-Task Cascaded Convolutional Neural Networks* adalah jaringan saraf yang digunakan untuk mendeteksi wajah dan *landmark* wajah pada

gambar. MTCNN diterbitkan pada tahun 2016 oleh Zhang et al [1]. MTCNN adalah suatu algoritma yang terdiri dari 3 jaringan saraf konvolasional, yaitu P-Net, R-Net, dan O-Net, yang mendeteksi *bounding box* pada wajah dan 5 *Point Face Landmarks* dalam sebuah gambar. Secara bertahap, setiap jaringan saraf meningkatkan hasil deteksi dengan melewatkan inputnya melalui CNN dan diikuti oleh *Non-Maximum Suppression*, atau NMS, yaitu metode yang mengurangi jumlah *bounding box* [2].

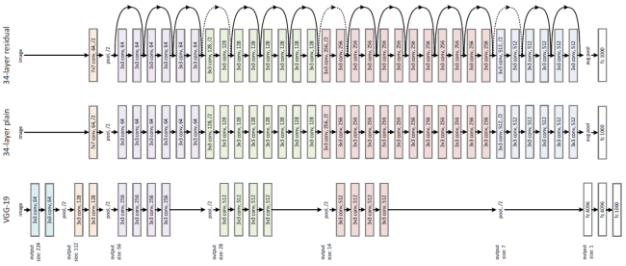
B. FaceNet




**Gambar 2** Arsitektur FaceNet

FaceNet adalah *deep neural network* yang digunakan untuk mengekstraksi fitur dari gambar wajah seseorang. FaceNet diterbitkan pada tahun 2015 oleh peneliti Google Schroff et al [3]. FaceNet adalah sebuah *neural network* yang memetakan wajah seseorang menjadi *Euclidean space* (kumpulan dari *geometrical point*) yang nantinya *geometrical point* tersebut menentukan nilai untuk mengukur tingkat kemiripan wajah. Sehingga jika wajah tersebut semakin mirip akan membuat nilainya semakin kecil dan kebalikannya jika wajah tersebut semakin berbeda akan menghasilkan nilai yang semakin besar [4].

C. ResNet50

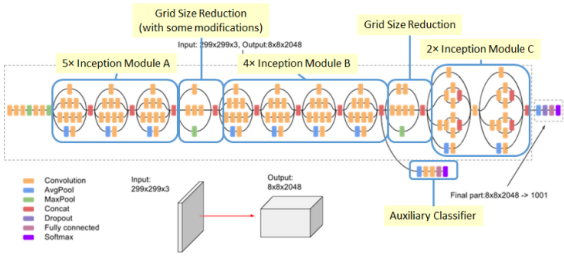


**Gambar 3** Arsitektur ResNet50

Residual Network (ResNet) memperkenalkan konsep *Skip Connection* atau *Shortcut Connection* untuk meneruskan input dari layer sebelumnya menuju ke layer berikutnya tanpa mengubah nilai input-nya.

ResNet sejatinya memiliki arsitektur yang identik dengan *Deep Learning* biasa, yang membedakan hanyalah ResNet menambahkan Skip/Shortcut Connection untuk mengurangi dampak dari Vanishing/Exploding Gradient pada jumlah layer yang massif [5].

D. InceptionV3

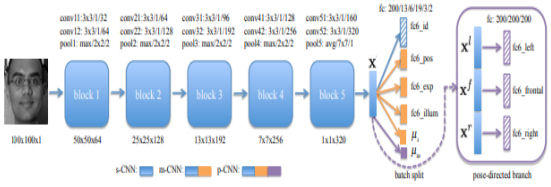



**Gambar 4** Arsitektur InceptionV3

InceptionV3 terdiri dari 42 layer yang terdiri dari tiga modul utama. Modul A menggunakan Factorization into Smaller Convolutions, yang mana menggantikan sebuah 5x5 filter dari generasi sebelumnya dengan dua buah 3x3 filters sehingga mengurangi jumlah parameter sebesar 28%. Modul B menggunakan Factorization into Asymmetric Convolutions yang disusun secara seri untuk mengganti sebuah 3x3 filter dengan dua buah filter asimetris, yakni 3x1 dan 1x3 filters. Modul ini berhasil mengurangi jumlah parameter sebanyak 33% dari generasi sebelumnya. Modul C juga menggunakan Factorization into Asymmetric Convolutions, namun disusun secara paralel, yang menghasilkan output dengan dimensi yang lebih banyak Setiap modul pada InceptionV3 dijembatani oleh Auxiliary Classifier supaya model dapat menyaring lebih banyak ciri. Terdapat Grid Size Reduction di antara Modul A dengan B, dan Modul B dengan C. Grid Size Reduction ini memiliki fungsi yang identik dengan lapisan Pooling pada arsitektur lain, yaitu mereduksi nilai dari Feature Map Matrix ke dalam dimensi yang lebih kecil [6].

E. Pose Invariant Model

Pose invariant model adalah suatu arsitektur jaringan saraf yang dibangun oleh Xi Yin dan Xiaoming Liu. Model ini dibangun untuk melakukan deteksi pengenalan wajah yang mana gambar wajah tersedia dalam 3 variasi pose atau sudut pandang, yaitu tampak depan, tampak samping kiri, dan tampak samping kanan. Model ini memanfaatkan *multi-task learning* (MTL) untuk metode pengenalan wajah. Berikut gambar arsitektur dari model ini [7].




**Gambar 5** Arsitektur Pose Invariant

F. ResNet Dorian Lazar

Pada penelitian atau tugas ini, kami juga menggukan model ResNet yang dibangun oleh Dorian Lazar. ResNet milik Dorian Lazar ini mengunakan CIFAR-10 sebagai *dataset*-nya. ResNet dibangun dengan menggunakan input *shape* (32, 32, 3) dan 10 output *node*, dalam hal ini kami mengubah input *shape* menjadi (224, 224, 3) dan 5 output *node*. Model ini memiliki total 30 *convolutional* dan *dense layer*. Semua ukuran

kernel adalah 3x3, serta menggunakan aktivasi ReLU dan BatchNormalisasi setelah *convolutional layer* [8].

## II. METODE PENELITIAN

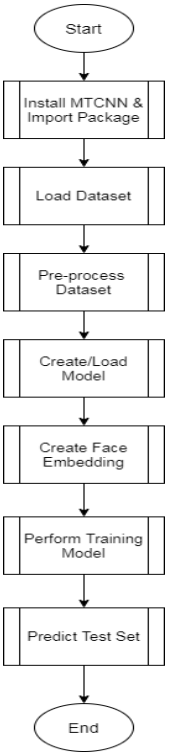
### A. Data Penelitian

Data yang digunakan pada penelitian/tugas ini merupakan data yang berasal dari *Extended Yale Face Database B*. *Extended Yale Face Database B* adalah *database* yang berisi 16128 gambar dari 28 subyek manusia pada 9 pose dan 64 kondisi pencahayaan. Dalam penelitian ini, kami menggunakan 5 buah *sub-database* (*folder*), yaitu YaleB26, YaleB27, YaleB28, YaleB29, dan YaleB30, yang semuanya berjumlah 2925 data gambar dalam format pgm.

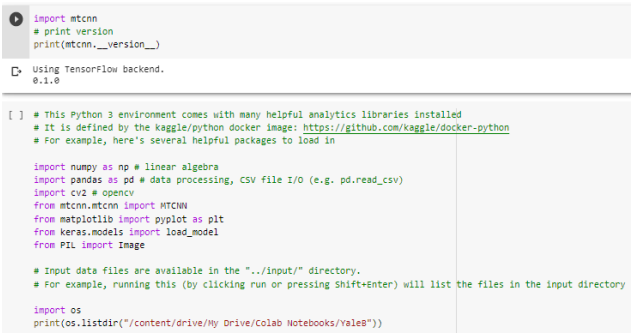
**Gambar 6** Sampel Dataset Yale B

### B. Langkah Penelitian

Dengan 5 arsitektur jaringan saraf yang kami gunakan artinya terdapat 5 *notebook python* yang kami buat. Kelima *notebook* tersebut pada dasarnya memiliki struktur kode program yang sama hanya terdapat perbedaan pada beberapa bagian saja. Berikut langkah yang digunakan untuk melakukan penelitian/tugas ini.




**Gambar 7** Diagram Alir Langkah Penelitian



```
import mtcnn
# print version
print(mtcnn.__version__)
```

```
Using TensorFlow backend.
0.1.0
```

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import cv2 # opencv
from mtcnn.mtcnn import MTCNN
from matplotlib import pyplot as plt
from keras.models import load_model
from PIL import Image

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

import os
print(os.listdir("/content/drive/My Drive/Colab Notebooks/YaleB"))
```

**Gambar 8** *Install* MTCNN & *Import Package*

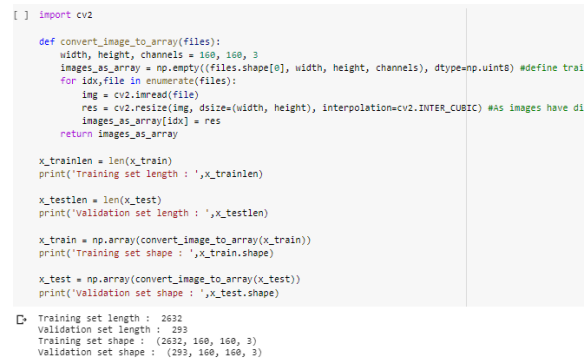
```
from sklearn.datasets import load_files
import numpy as np

data_dir = '/content/drive/My Drive/Colab Notebooks/YaleB/'

def load_dataset(path):
    data = load_files(path) #load all files from the path
    files = np.array(data['filenames']) #get the file
    targets = np.array(data['target'])#get the the classification labels as integer index
    target_labels = np.array(data['target_names'])#get the the classification labels
    return files,targets,target_labels

x_data, y_data,target_labels = load_dataset(data_dir)
print('Dataset size : ' , x_data.shape[0])
```

```
Dataset size :  2925
```

```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(x_data,y_data,test_size = 0.1,random_state = 1)

print ("x_train shape: " + str(x_train.shape))
print ("y_train shape: " + str(y_train.shape))
print ("x_test shape: " + str(x_test.shape))
print ("y_test shape: " + str(y_test.shape))
```

```
x_train shape: (2632,)
y_train shape: (2632,)
x_test shape: (293,)
y_test shape: (293,)
```

**Gambar 9** *Load* Dataset



```
import cv2

def convert_image_to_array(files):
    width, height, channels = 160, 160, 3
    images_as_array = np.empty((files.shape[0], width, height, channels), dtype=np.uint8) #define trai
    for idx,file in enumerate(files):
        img = cv2.imread(file)
        res = cv2.resize(img, dsize=(width, height), interpolation=cv2.INTER_CUBIC) #As images have di
        images_as_array[idx] = res
    return images_as_array

x_trainlen = len(x_train)
print('Training set length : ',x_trainlen)

x_testlen = len(x_test)
print('Validation set length : ',x_testlen)

x_train = np.array(convert_image_to_array(x_train))
print('Training set shape : ',x_train.shape)

x_test = np.array(convert_image_to_array(x_test))
print('Validation set shape : ',x_test.shape)
```

```
Training set length :  2632
Validation set length :  293
Training set shape :  (2632, 160, 160, 3)
Validation set shape :  (293, 160, 160, 3)
```

**Gambar 10** *Pre-process Data*

Pertama-tama dilakukan instalasi MTCNN dan pengimporan package yang diperlukan, seperti numpy, pandas, pyplot, dll. Selanjutnya, dilakukan mekanisme *load* dan *pre-process dataset* yang diawali proses pemuatan dataset ke runtime, lalu *split* atau pembagian dataset menjadi data *train* dan data *test*. Di bagian ini, kami melakukan 2 variasi *split* data, yang nantinya akan dibandingkan hasilnya, yaitu 90% data *train* dan 10% data *test*, serta 80% data *train* dan 20% data *test*. Terakhir, mengkonversi data gambar ke dalam bentuk *array*.

```
[ ] from tensorflow import Tensor
    from tensorflow.keras.layers import Input, Conv2D, ReLU, BatchNormalization,\
                                        Add, AveragePooling2D, Flatten, Dense
    from tensorflow.keras.models import Model

    def relu_bn(inputs: Tensor) -> Tensor:
        relu = ReLU()(inputs)
        bn = BatchNormalization()(relu)
        return bn

    def residual_block(x: Tensor, downsample: bool, filters: int, kernel_size: int = 3) -> Tensor:
        y = Conv2D(kernel_size=kernel_size,
                   strides= (1 if not downsample else 2),
                   filters=filters,
                   padding="same")(x)
        y = relu_bn(y)
        y = Conv2D(kernel_size=kernel_size,
                   strides=1,
                   filters=filters,
                   padding="same")(y)

        if downsample:
            x = Conv2D(kernel_size=1,
                       strides=2,
                       filters=filters,
                       padding="same")(x)
        out = Add()([x, y])
        out = relu_bn(out)
        return out

    def create_res_net():

        inputs = Input(shape=(224, 224, 3))
        num_filters = 64

        t = BatchNormalization()(inputs)
        t = Conv2D(kernel_size=3,
                   strides=1,
                   filters=num_filters,
                   padding="same")(t)
```

**Gambar 11** *Create* Model

Berikutnya, tahap pembentukan model jaringan saraf, yaitu tahap dimana dilakukan penentuan *layer-layer* jaringan saraf beserta parameternya, penentuan jumlah layer dan susunan *layer*-nya, penentuan fungsi aktivasi, dan penentuan *optimizer* serta loss function-nya, yang nantinya jaringan saraf ini digunakan untuk melakukan *training* data. Khusus untuk *pre-trained* model, kita hanya perlu memuat model tersebut ke dalam *runtime*.

```
[ ] def get_embedding(model, face):
        # scale pixel values
        face = face.astype('float32')
        # standardization
        mean, std = face.mean(), face.std()
        face = (face-mean)/std
        # transfer face into one sample (3 dimension to 4 dimension)
        sample = np.expand_dims(face, axis=0)
        # make prediction to get embedding
        yhat = model.predict(sample)
        return yhat[0]

    # convert each face in the train set into embedding
    emdTrainX = list()
    for face in x_train:
        emd = get_embedding(facenet_model, face)
        emdTrainX.append(emd)

    emdTrainX = np.asarray(emdTrainX)
    print(emdTrainX.shape)

    # convert each face in the test set into embedding
    emdTestX = list()
    for face in x_test:
        emd = get_embedding(facenet_model, face)
        emdTestX.append(emd)
    emdTestX = np.asarray(emdTestX)
    print(emdTestX.shape)

    (2632, 128)
    (293, 128)
```

**Gambar 12** *Create Face Embbeding*

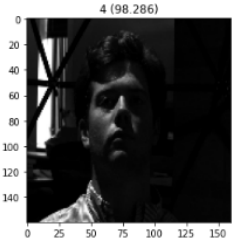
Tahap selanjutnya adalah membuat *face embedding*. *Face embedding* adalah vektor numerik yang merepresentasikan fitur-fitur yang diekstrak dari gambar wajah. Vektor ini kemudian dapat dibandingkan dengan vektor yang dihasilkan dari gambar wajah lain. Misalnya, vektor dari satu gambar dengan gambar yang lain berdekatan maka kemungkinan gambar-gambar tersebut menunjukan orang yang sama, sedangkan vektor satu gambar dengan gambar yang lain berbeda jauh maka kemungkinan gambar-gambar tadi dari orang yang berbeda. Proses pembentukan *face embedding* ini memanfaatkan model yang sudah kita buat di tahap sebelumnya tadi. Proses pembentukan *face embedding* dari suatu gambar meliputi normalisasi, standarisasi, dan ekspansi dimensi gambar wajah sehingga diperoleh *embedding*-nya. Setiap data gambar pada *train* dan *test* set dilakukan proses *face embedding*, sedemikian sehingga, *train* dan *test* set akan dikonversikan ke dalam bentuk vektor numerik *face embedding*.

```
[ ] from sklearn.metrics import accuracy_score
    from sklearn.preprocessing import LabelEncoder
    from sklearn.preprocessing import Normalizer
    from sklearn.svm import SVC

    print("Dataset: train=%d, test=%d" % (emdTrainX.shape[0], emdTestX.shape[0]))
    # normalize input vectors
    in_encoder = Normalizer()
    emdTrainX_norm = in_encoder.transform(emdTrainX)
    emdTestX_norm = in_encoder.transform(emdTestX)
    # label encode targets
    out_encoder = LabelEncoder()
    out_encoder.fit(y_train)
    trainy_enc = out_encoder.transform(y_train)
    testy_enc = out_encoder.transform(y_test)
    # fit model
    model = SVC(kernel='linear', probability=True)
    model.fit(emdTrainX_norm, trainy_enc)
    # predict
    yhat_train = model.predict(emdTrainX_norm)
    yhat_test = model.predict(emdTestX_norm)
    # score
    score_train = accuracy_score(trainy_enc, yhat_train)
    score_test = accuracy_score(testy_enc, yhat_test)
    # summarize
    print('Accuracy: train=%.3f, test=%.3f' % (score_train*100, score_test*100))

    Dataset: train=2632, test=293
    Accuracy: train=96.201, test=96.587
```

**Gambar 13** *Perform Training Model*

Tahap *training* model dilakukan bukan dengan model yang kita buat sebelumnya, melainkan model yang dibangun dengan memanfaatkan *Support Vector Machine* (SVM), yaitu *Support Vector Classification* (SVC). Perlu diingat bahwa pada tahap ini, input data *train* dan *test* bukan lagi berupa *array* melainkan berbentuk vektor hasil dari *face embedding*. Maka dari itu digunakan algoritma SVM yang memang efektif untuk klasifikasi vektor *face embedding*. Pada tahap ini, dilakukan normalisasi vektor, lalu *encode* label. Barulah dilakukan *training* model SVC dan diperoleh akurasi data *train* dan *test*. Kami melakukan percobaan dengan 4 variasi kernel SVM, yaitu *linear*, *sigmoid*, *Radial Basis Function (RBF)*, dan *polynomial*.

```
    yhat_prob = model.predict_proba(samples)
    # get name
    class_index = yhat_class[0]
    class_probability = yhat_prob[0,class_index] * 100
    predict_names = out_encoder.inverse_transform(yhat_class)
    all_names = out_encoder.inverse_transform([0,1,2,3,4])
    #print('Predicted: %s (%.3f)' % (predict_names[0], class_probability))
    print('Predicted: \n%s \n%s' % (all_names, yhat_prob[0]*100))
    print('Expected: %s' % random_face_name[0])
    # plot face
    plt.imshow(random_face)
    title = '%s (%.3f)' % (predict_names[0], class_probability)
    plt.title(title)
    plt.show()

Predicted:
[0 1 2 3 4]
[1.22494277e+00 3.02418600e-01 1.04831332e-02 1.75897817e-01
 9.82862577e+01]
Expected: 4
```




**Gambar 14** *Predict Test Set*

Tahap terakhir adalah memprediksi gambar acak. Pada tahap ini, kita perlu mengambil sampel acak dari *test* set, kemudian mendapatkan *face embedding*-nya, piksel wajah, prediksi kelas yang diharapkan, dan nama/label yang sesuai untuk kelas tersebut.

Selanjutnya, kita dapat menggunakan face embedding sebagai input untuk membuat sebuah prediksi dengan fit model.

C. Variabel Terikat dan Bebas

Variabel terikat yang digunakan pada seluruh model dirangkum sebagai berikut:

Jumlah Gambar : 2925
Model Fit : SVM
Pengkonversian Input : Face Embedding

Variabel terikat yang khusus digunakan pada sekuensial model dirangkum sebagai berikut:

Optimizer : Adam
Learning Rate : 0.001
Metric : *Accuracy*
Loss function : *Categorical Crossentropy*

Variabel bebas yang dibandingkan dapat dirinci sebagai berkut:

Arsitektur : FaceNet, ResNet Dorian Lazar, Pose Invariant, ResNet50, InceptionV3
*Split* data (*train* - *test*) : 90% - 10%; 80% - 20%
Kernel SVM : Linear, Sigmoid, RBF, Polynomial

## III. HASIL DAN PEMBAHASAN

A. FaceNet

Pada model FaceNet, dengan resolusi input adalah 160x160 *pixel*, dilakukan 7 kali percobaan dengan 2 variasi split data (*train – test*) dan 4 variasi kernel SVM. Hasil yang diperoleh menunjukan bahwa *split* data dengan 90% data *train* dan 10% data *test* dengan kernel *polynomial* memiliki akurasi terbaik, yaitu 99.24% untuk *train* dan 98.29% untuk *test*.

**Tabel 1** Hasil Percobaan Model FaceNet

| Model | Split data (*train – test*) | Kernel SVM | Accuracy *train* (%) | Accuracy *test* (%) |
|---|---|---|---|---|
| FaceNet | 90% - 10% | Linear | 96.2 | 96.58 |
| | | Sigmoid | 83.55 | 83.96 |
| | | RBF | 98.74 | 98.3 |
| | | **Polynomial** | **99.24** | **98.29** |
| | 80% - 20% | Linear | 96.07 | 95.21 |
| | | RBF | 98.55 | 97.78 |
| | | Polynomial | 99.19 | 98.63 |

B. ResNet Dorian Lazar

Pada model ResNet Dorian Lazar, dengan resolusi input adalah 224x224 *pixel*, dilakukan 7 kali percobaan dengan 2 variasi split data (*train – test*) dan 4 variasi kernel SVM. Hasil yang diperoleh menunjukan bahwa *split* data dengan 90% data *train* dan 10% data *test* dengan kernel *polynomial* memiliki akurasi terbaik, yaitu 69.26% untuk *train* dan 69.97% untuk *test*.

**Tabel 2** Hasil Percobaan Model ResNet Dorian Lazar

| Model | Split data (*train – test*) | Kernel SVM | Accuracy *train* (%) | Accuracy *test* (%) |
|---|---|---|---|---|
| ResNet Dorian Lazar | 90% - 10% | Linear | 44.64 | 45.05 |
| | | Sigmoid | 20.67 | 13.99 |
| | | RBF | 65.39 | 64.5 |
| | | **Polynomial** | **69.26** | **69.97** |
| | 80% - 20% | Linear | 52.43 | 54.53 |
| | | Polynomial | 67.52 | 67 |
| | | RBF | 63.63 | 63.76 |

C. Pose Invariant

Pada *model Pose Invariant*, dengan resolusi input adalah 160x160 *pixel*, dilakukan 7 kali percobaan dengan 2 variasi split data (*train – test*) dan 4 variasi kernel SVM. Hasil yang diperoleh menunjukan bahwa *split* data dengan 90% data *train* dan 10% data *test* dengan kernel *Radial Basis Function (RBF)* memiliki akurasi terbaik, yaitu 51.6% untuk *train* dan 47.78% untuk *test*.

**Tabel 3** Hasil Percobaan Model Pose Invariant

| Model | Split data (*train – test*) | Kernel SVM | Accuracy *train* (%) | Accuracy *test* (%) |
|---|---|---|---|---|
| Pose Invariant | 90% - 10% | Linear | 20.67 | 13.99 |
| | | RBF | 51.6 | 47.78 |
| | | Polynomial | 49.62 | 46.08 |
| | 80% - 20% | Linear | 21.03 | 15.9 |
| | | Sigmoid | 21.03 | 15.9 |
| | | **RBF** | **57.35** | **56.58** |
| | | Polynomial | 51.37 | 49.92 |

D. ResNet50

Pada *model* ResNet50, dengan resolusi input adalah 224x224 *pixel* dan bobot menggunakan ImageNet, dilakukan 7 kali percobaan dengan 2 variasi split data (*train – test*) dan 4 variasi kernel SVM. Hasil yang diperoleh menunjukan bahwa *split* data dengan 90% data *train* dan 10% data *test* dengan kernel *polynomial* memiliki akurasi terbaik, yaitu 98.4% untuk *train* dan 97.95% untuk *test*.

**Tabel 4** Hasil Percobaan Model ResNet50

| Model | Split data (*train – test*) | Kernel SVM | Accuracy *train* (%) | Accuracy *test* (%) |
|---|---|---|---|---|
| ResNet50 | 90% - 10% | Linear | 95.06 | 96.56 |
| | | Sigmoid | 88.45 | 91.13 |
| | | RBF | 97.23 | 97.61 |
| | | **Polynomial** | **98.4** | **97.95** |
| | 80% - | Linear | 94.78 | 95.04 |

| | | | | |
|---|---|---|---|---|
| | 20% | RBF | 96.97 | 97.44 |
| | | Polynomial | 98.21 | 98.46 |

E. InceptionV3

Pada *model* InceptionV3, dengan resolusi input adalah 299x299 *pixel* dan bobot menggunakan ImageNet, dilakukan 7 kali percobaan dengan 2 variasi split data (*train – test*) dan 4 variasi kernel SVM. Hasil yang diperoleh menunjukan bahwa *split* data dengan 90% data *train* dan 10% data *test* dengan kernel *polynomial* memiliki akurasi terbaik, yaitu 76.59% untuk *train* dan 77.13% untuk *test*.

**Tabel 5** Hasil Percobaan Model InceptionV3

| Model | Split data (*train – test*) | Kernel SVM | Accuracy *train* (%) | Accuracy *test* (%) |
|---|---|---|---|---|
| InceptionV3 | 90% - 10% | Linear | 70.48 | 75.43 |
| | | RBF | 75.95 | 76.45 |
| | | **Polynomial** | **76.59** | **77.13** |
| | 80% - 20% | Linear | 70.1 | 72.99 |
| | | Sigmoid | 60.64 | 66.84 |
| | | RBF | 75.299 | 74.188 |
| | | Polynomial | 75.86 | 74.02 |

F. Keseluruhan

Dari percobaan-percobaan yang dilakukan kelima model tersebut, kernel *polynomial* dan RBF memiliki akurasi yang paling baik. Sedangkan, akurasi terburuk selalu diperoleh saat menggunkan kernel *sigmoid*. Pada model FaceNet, ResNet DL (Dorian Lazar), ResNet50, dan InceptioV3 akurasi terbaik didapatkan dengan menggunakan kernel *polynomial*. Sedangkan untuk model Pose Invariant akurasi terbaiknya saat menggunakan kernel RBF. Berikut tabel konfigurasi terbaik untuk setiap model:

**Tabel 6** Konfigurasi Terbaik Setiap Arsitektur

| Model | Split data (*train – test*) | Kernel SVM | Accuracy *train* (%) | Accuracy *test* (%) |
|---|---|---|---|---|
| FaceNet | 90% - 10% | Polynomial | 99.24 | 98.29 |
| ResNet DL | 90% - 10% | Polynomial | 69.26 | 69.97 |
| Pose Invariant | 80% - 20% | RBF | 57.35 | 56.58 |
| ResNet50 | 90% - 10% | Polynomial | 98.4 | 97.95 |
| InceptionV3 | 90% - 10% | Polynomial | 76.59 | 77.13 |

## IV. KESIMPULAN DAN SARAN

A. Kesimpulan

1. Model terbaik untuk pengenalan wajah dengan variasi pose dan kondisi pencahayaan adalah FaceNet dan ResNet50. Dengan konfigurasi untuk kedua model tersebut adalah split data *train* dan *test* 90% - 10% dan kernel SVM adalah polynomial, tingkat akurasinya mencapai 98%. Khusus ResNet50, model tersebut menggunakan weight ImageNet.
2. Face Embedding digunakan untuk mengkonversikan input gambar (*array*) menjadi vektor numerik yang nantinya digunakan untuk mempermudah dalam mengklasifikasikan gambar wajah.
3. *Support Vector Machine* (SVM) digunakan untuk melakukan pengklasifikasian input gambar yang sudah berupa vektor numeric hasil *face embbeding*.
4. Kernel SVM terbaik yang digunakan untuk pengenalan wajah dengan variasi pose dan kondisi pencahayaan adalah *polynomial* dan *Radial Basis Function* (RBF).
5. Kernel SVM paling tidak cocok untuk pengenalan wajah dengan variasi pose dan kondisi pencahayaan adalah *sigmoid*.

B. Saran

1. Menggunakan dataset yang jauh lebih besar lagi, dengan komposisi data yang lebih baik, serta sampel wajah yang lebih beragam.
2. Membandingkan metode dan arsitektur lain dengan kombinasi konfigurasi yang lebih beragam sehingga bisa mendapatkan hasil prediksi yang lebih akurat.
3. Dapat memvisalisasikan data uji sehingga kita dapat lebih memahami karakteristik data, seperti persebaran data dalam bentuk histogram, persebaran data dalam bentuk PCA, persebaran data prediksi dalam bentuk *confusion matrix*, dan bentuk-bentuk lainnya.

## DAFTAR PUSTAKA


[1] C.-F. Wang, "How Does A Face Detection Program Work? (Using Neural Network)," towards data science, 27 Juli 2018. [Online]. Available: https://towardsdatascience.com/how-does-a-face-detection-program-work-using-neural-networks-17896df8e6ff. [Accessed 10 Juni 2020].

[2] V. Muhler, "Realtime JavaScript Face Tracking and Face Recognition using face-api.js' MTCNN Face Detector," ITNEXT, 16 Juli 2018. [Online]. Available: https://itnext.io/realtime-javascript-face-tracking-and-face-recognition-using-face-api-js-mtcnn-face-detector-d924dd8b5740. [Accessed 10 Juni 2020].

[3] L. Dulčić, "Face Recognition with FaceNet and MTCNN," arsfutura, Desember 2019. [Online]. Available: https://arsfutura.com/magazine/face-recognition-with-facenet-and-mtcnn/. [Accessed 10 Juni 2020].

[4] S. Skúli, "Making your own Face Recognition System," freeCodeCamp, 11 Januari 2018. [Online]. Available:

https://www.freecodecamp.org/news/making-your-own-face-recognition-system-29a8e728107c/. [Accessed 10 Juni 2020].

[5] S.-H. Tsang, "Review: ResNet — Winner of ILSVRC 2015 (Image Classification, Localization, Detection)," towards data science, 15 September 2018. [Online]. Available: https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8. [Accessed 11 Juni 2020].

[6] S.-H. Tsang, "Review: Inception-v3 — 1st Runner Up (Image Classification) in ILSVRC 2015," medium.com, 10 September 2018. [Online]. Available: https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c. [Accessed 11 Juni 2020].

[7] X. Yin and X. Liu, "Multi-Task Convolutional Neural Network for Pose-Invariant Face Recognition," *IEEE Transactions on Image Processing*, vol. 27, no. 2, pp. 964-975, 2017.

[8] D. Lazar, "Building a ResNet in Keras," towards data science, 6 Maret 2020. [Online]. Available: https://towardsdatascience.com/building-a-resnet-in-keras-e8f1322a49ba. [Accessed 11 Juni 2020].