4. IMPLEMENTASI SISTEM

Pada bab ini akan dibahas mengenai implementasi sesuai analisa kebutuhan dengan desain sistem yang dibuat pada bab sebelumnya. Implementasi sistem ini meliputi instalasi *library*, pengambilan data, pengolahan data, implementasi *training,* dan *testing* sistem, dan implementasi aplikasi.

Tabel 4.1 Segmen implementasi sistem

Segmen	Proses	Flowchart	Keterangan
Segmen Program 4.4	*Build Makefile*	-	Melakukan build pada Makefile untuk training dan testing YOLOv4 dan YOLOv4-tiny
Segmen Program 4.5	Implementasi *Training YOLOv4*	3.10	Melakukan training menggunakan YOLOv4
Segmen Program 4.6	Implementasi *Training YOLOv4-tiny*	3.11	Melakukan training menggunakan YOLOv4-tiny
Segmen Program 4.8	*Build* Arsitektur AlexNet	-	Melakukan *build* pada arsitektur AlexNet agar dapat dilakukan *train* dan *test*
Segmen Program 4.12	Implementasi *Training AlexNet*	3.13	Melakukan training menggunakan AlexNet
Segmen Program 4.14	Implementasi *Testing YOLOv4*	3.12	Melakukan testing pada hasil training yang telah dilakukan oleh YOLOv4 DarkNet
Segmen Program 4.15	Implementasi *Testing YOLOv4-tiny*	3.12	Melakukan testing pada hasil training yang telah dilakukan oleh YOLOv4-tiny
Segmen Program 4.16	Implementasi *Testing AlexNet*	3.14	Melakukan testing pada hasil training yang telah dilakukan oleh AlexNet
Segmen Program 4.18	Implementasi Aplikasi	-	Implementasi Aplikasi pada *desktop*

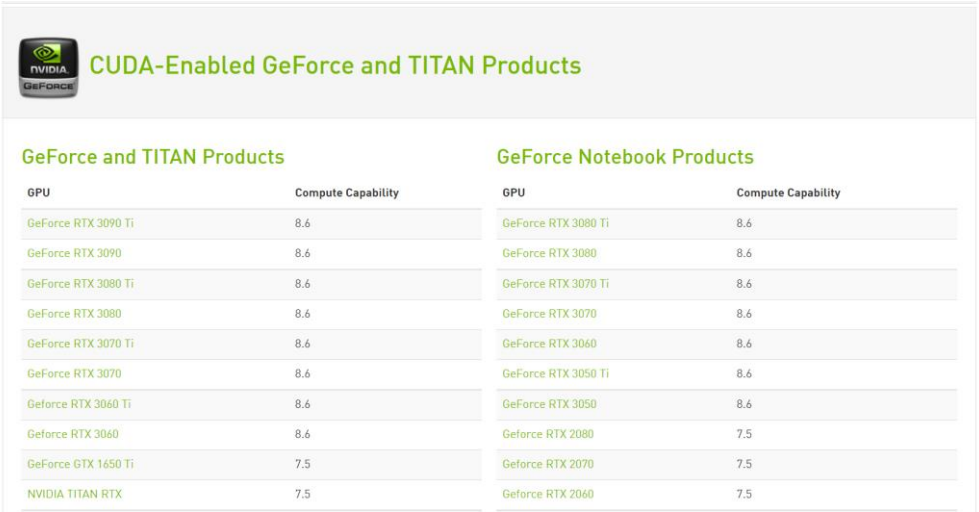
4.1. Instalasi dan konfigurasi Library

Dalam tahap ini, diperlukan untuk melakukan instalasi dan konfigurasi *Tensorflow*, dan *Keras* terlebih dahulu. Instalasi akan dilakukan pada laptop menggunakan sistem operasi *Windows 11* dengan spesifikasi GPU: *Nvidia GeForce GTX 1650 4 GB.* Penelitian akan dilakukan menggunakan *Google Colab* sebagai tempat *training* dan *testing*. Pada penelitian ini perlu dipersiapkan hal-hal berikut:

1. Python 3.10.7
2. CUDA 10.1
3. cuDNN 7.6.5
4. Tensorflow 2.10.0
5. Keras

Pertama diperlukan untuk instalasi CUDA, dan cuDNN. CUDA, dan cuDNN berfungsi agar pada saat program dijalankan, program akan menggunakan *GPU (Graphics Processing Unit)* sehingga proses akan berjalan dengan lebih cepat jika dibandingkan dengan *CPU* / *System RAM (Random Access Memory)*. Digunakan versi 10.1 untuk instalasi CUDA *toolkit*, dan versi 7.6.5 untuk cuDNN. Berikut adalah tahap instalasi CUDA, dan cuDNN.

1. Karena setiap GPU memiliki *compatibility* yang berbeda-beda untuk setiap versi CUDA, maka harus dipastikan terlebih dahulu apakah CUDA dapat berjalan pada GPU yang digunakan. Pengecekan dapat dilihat dari https://developer.nvidia.com/cuda-gpus seperti yang terlihat pada Gambar 4.1



CUDA-Enabled GeForce and TITAN Products

GeForce and TITAN Products

GPU	Compute Capability
GeForce RTX 3090 Ti	8.6
GeForce RTX 3090	8.6
GeForce RTX 3080 Ti	8.6
GeForce RTX 3080	8.6
GeForce RTX 3070 Ti	8.6
GeForce RTX 3070	8.6
Geforce RTX 3060 Ti	8.6
Geforce RTX 3060	8.6
GeForce GTX 1650 Ti	7.5
NVIDIA TITAN RTX	7.5

GeForce Notebook Products

GPU	Compute Capability
GeForce RTX 3080 Ti	8.6
GeForce RTX 3080	8.6
GeForce RTX 3070 Ti	8.6
GeForce RTX 3070	8.6
GeForce RTX 3060	8.6
GeForce RTX 3050 Ti	8.6
GeForce RTX 3050	8.6
Geforce RTX 2080	7.5
Geforce RTX 2070	7.5
Geforce RTX 2060	7.5

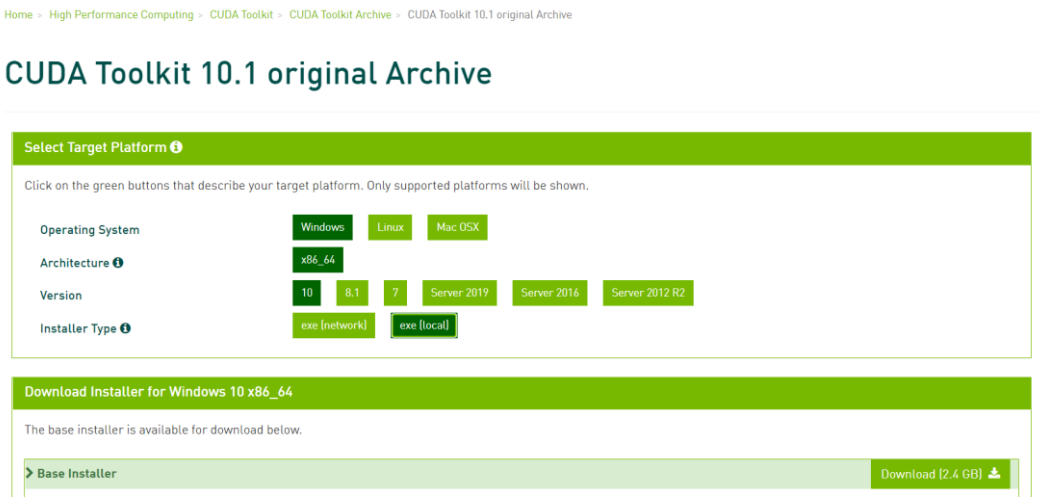
Gambar 4.1 Daftar NVIDIA GPU yang *compatible* dengan CUDA

2. Setelah dipastikan bahwa GPU yang digunakan *compatible* dengan CUDA, maka hal yang selanjutnya perlu dilakukan adalah melakukan instalasi tensorflow menggunakan *command* pada Anaconda prompt. *Command* dapat dilihat pada Gambar 4.2

```
(coba) C:\Users\brico>pip install tensorflow
```

Gambar 4.2 *Command* untuk instalasi tensorflow

3. Setelah tensorflow telah terinstall, maka hal selanjutnya adalah mengunduh CUDA dari https://developer.nvidia.com/cuda-10.1-download-archive-base seperti pada Gambar 4.3

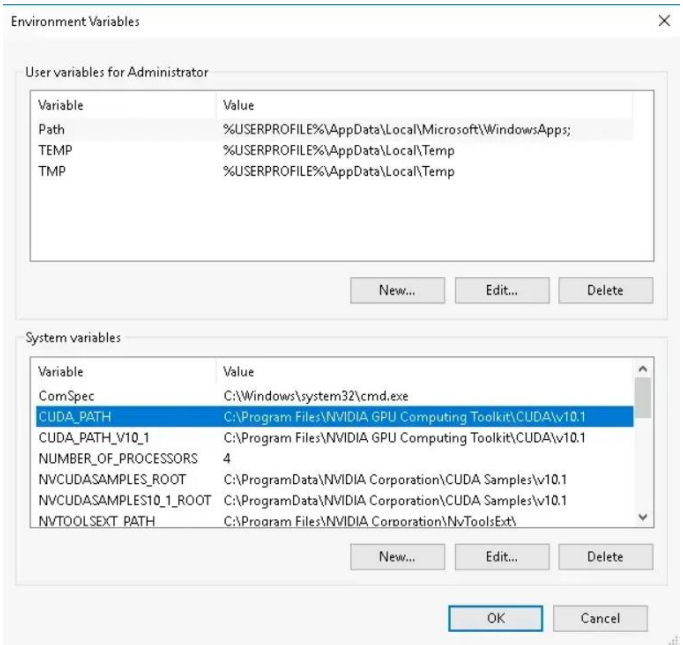


Gambar 4.3 Download CUDA dengan versi 10.1

4. Apabila CUDA sudah terunduh dan sudah di install, maka selanjutnya adalah melakukan pengunduhan, dan menginstall cuDNN melalui *website* NVIDIA yang memiliki alamat https://developer.nvidia.com/rdp/cudnn-archive. Pengunduhan dapat dilihat pada Gambar 4.4

Gambar 4.4 Download cuDNN versi 7.6.5

5. Lalu selanjutnya yang harus dilakukan adalah melakukan ekstraksi file cuDNN ke dalam tempat instalasi CUDA, dan menambahkan variabel CUDA, dan cuDNN pada PATH *system environment* seperti yang terlihat pada Gambar 4.5



Gambar 4.5 menambahkan CUDA_PATH variabel pada system environment

Setelah instalasi CUDA, dan cuDNN selesai, maka langkah selanjutnya yang dilakukan adalah melakukan instalasi *library* pada python. Berikut adalah *library* yang digunakan dalam pengerjaan skripsi ini:

1. numpy
2. matplotlib

3. opencv-python
4. tk

Untuk instalasi *library* yang terlah disebutkan sebelumnya dapat dilakukan dengan menggunakan *command* dari *Anaconda Prompt*. *Command* yang digunakan untuk instalasi adalah: pip install <*lib_name*>, sehingga untuk instalasi numpy adalah: pip install numpy

4.2. Pengolahan Dataset

Pada tahap ini akan dijelaskan tentang proses pengolahan dataset yang dilakukan

4.2.1. Crop Dataset

Pada pemotongan tiap karakter Aksara Jawa, digunakan program python yang dapat memisahkan karakter Aksara Jawa berdasarkan anotasi yang diberi dengan cara melihat kedalam tiap *YOLO text file* yang telah dibuat pada saat proses anotasi gambar yang digunakan menggunakan *tools labelimg.* Cara kerja dari program ini adalah, pertama akan dibuat *dictionary* dari *classes.txt* yang berisi seluruh class dari *dataset* yang telah diberi anotasi dengan cara membaca tiap line dari *classes.txt* dan mengubahnya menjadi *dictionary*. Lalu program akan melakukan pengecekan pada satu folder yang berisi dataset yang terdiri dari gambar, dan *YOLO text file.* Kemudian untuk tiap gambar akan dilakukan perhitungan *width,* dan *height dimension* untuk mengukur dimensi gambar tersebut. Setelah itu, akan dicarikan juga *YOLO text file* untuk gambar tersebut, dan akan dilakukan *string_split* untuk tiap line dengan parameter spasi sehingga didapatkan *class_id, x_center, y_center, width*, dan *height* dari gambar tersebut. Setelah itu dilakukan *cropping image* menggunakan *img_crop,* dan proses penamaan menggunakan *imwrite*. Pada *dataset* yang digunakan akan diubah proses penamaan dari simbol titik dua (:) menjadi tulisan (titikdua) dikarenakan pada *Windows*, proses penamaan *file* tidak dapat dimasukkan karakter unik seperti titik dua (:). Proses untuk melakukan *crop* pada gambar berdasarkan label yang dimilikinya dapat dilihat pada Segmen Program 4.1

Segmen Program 4.1 *Crop image* berdasarkan labelnya

```
#create a dictionary
co2 = 0
dictionary = {}
with open("E:/SKRIPSI/done/All/classes.txt") as file:
for line in file:
```

```
      (key, value) = co2, line.strip()
      dictionary[int(key)] = value
      co2+=1

#Crop labelled image
import cv2

count = 0
for filenum in range(1, 130):
      img = cv2.imread("E:/SKRIPSI/done/All/" + str(filenum) + ".jpg")
      dh,dw, _ = img.shape
      print(dw,dh)

      file1 = open('E:/SKRIPSI/done/All/' + str(filenum) + '.txt',
      'r')

      for line in file1:
         count += 1
         box = "{}".format(line.strip())
         class_id, x_center, y_center, w, h = box.strip().split()
         x_center, y_center, w, h = float(x_center), float(y_center),
         float(w), float(h)
         x_center = round(x_center * dw)
         y_center = round(y_center * dh)
         w = round(w * dw)
         h = round(h * dh)
         x = round(x_center - w / 2)
         y = round(y_center - h / 2)

         imgCrop = img[y:y + h, x:x + w]
         if(int(class_id) == 19):
            cv2.imwrite('E:/SKRIPSI/done/All/cropped/' + 'titikdua' +
            '_' + str(count) + '.png',imgCrop)
         else:
            cv2.imwrite('E:/SKRIPSI/done/All/cropped/' +
            dictionary[int(class_id)] + '_' + str(count) +
            '.png',imgCrop)

      # Closing files
```

```
        file1.close()
        print('Count: ',count)

print('All image cropped!')
```

4.2.2. Proses Pelabelan Ulang

Setelah semua gambar telah terpotong dalam satu folder, maka dijalankan program python yang berfungsi untuk memberikan label pada gambar yang telah dipotong. Proses pelabelan ini menggunakan *classes.txt* sebagai pencocokan terhadap nama tiap gambar yang telah dipotong. *Value* x, y, *width*, dan *height* pada tiap label *text file* akan diubah menjadi 0.5, 0.5, 1, 1. *Command* untuk melakukan penambahan label untuk tiap gambar yang telah dipotong dapat dilihat pada Segmen Program 4.2 Berikut adalah cara perhitungan untuk isi *text file* YOLO:

- x_center = ((0 + max_width) / 2) / max_width
- y_center = ((0 + max_height) / 2) / max_height
- width = (max_width - 0) / max_width
- height = (max_height - 0) / max_height

Contoh dari YOLO *text file* dapat dilihat pada Gambar 4.6

```
41 0.337970 0.102260 0.021552 0.008295
```

Gambar 4.6 Contoh salah satu YOLO *text file*

Segmen Program 4.2 Menambahkan label untuk tiap gambar yang dipotong

```
#Creating yolo txt file from cropped files
import os
import re


for filename in os.scandir("train_txt"):
    if filename.is_file():
        filename = str(filename).split("'")
        x = filename[1].split("_")
        y = filename[1].split(".")
```

```
        with open('classes.txt') as myFile:
            for num, line in enumerate(myFile, 1):
                if re.search('^' + x[0] +'$', line):

                    with open(y[0] + '.txt', 'w') as f:
                            f.write( str(num-1) + ' 0.5 0.5 1 1')
                            print(str(filename[1]) + " written
successfully")

print("All file written succesfully!")
```

4.3. Implementasi Training

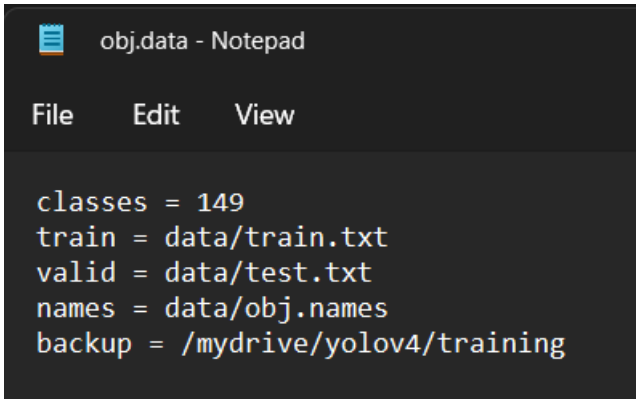
Pada segmen ini, akan dilakukan proses *training* yang akan dilakukan oleh YOLOv4 DarkNet, YOLOv4-tiny, dan AlexNet. Implementasi *training* sendiri akan dilakukan pada *Google Colab.*

4.3.1. Training YOLOv4 DarkNet

Pada proses *training* untuk YOLOv4 DarkNet akan dijelaskan proses persiapan data YOLOv4 sebelum dilakukan *training*, dan cara *training* menggunakan YOLOv4.

4.3.1.1. Persiapan Files

File-file yang perlu dipersiapkan sebelum melakukan *training* adalah file obj.names, obj.data, train.txt, dan test.txt. Berikut adalah isi dari tiap file yang telah telah disebutkan. Contoh isi obj.data dapat dilihat pada Gambar 4.7



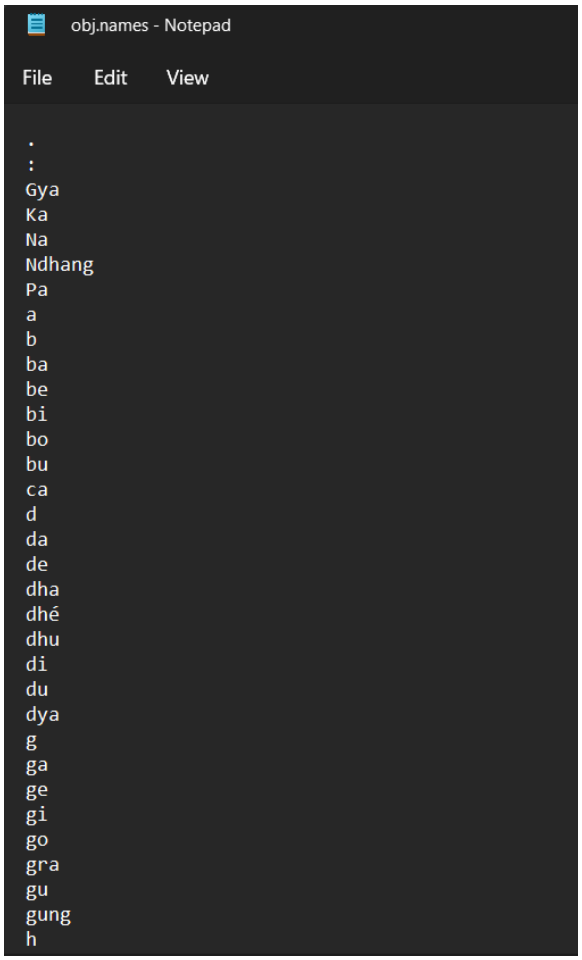
Gambar 4.7 isi file obj.data

Isi file dari obj.data dengan variable-variabel didalamnya dapat dijelaskan pada Tabel 4.2 berikut:

Tabel 4.2 Isi obj.data

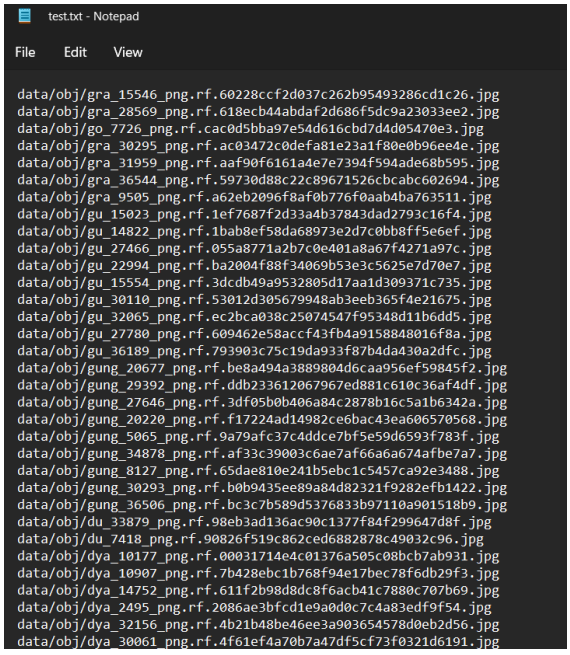
Nama	Deskripsi
classes	Jumlah class yang terdapat dalam dataset tersebut
train	Lokasi dari train.txt *file*
valid	Lokasi dari test.txt *file*
names	Lokasi dari obj.names *file*
backup	Lokasi dimana weight hasil *training* akan disimpan

Lalu untuk file obj.names memiliki isi berupa nama kelas yang terdapat dalam satu dataset seperti classes.txt. Contoh classes.txt dapat dilihat pada Gambar 4.8



Gambar 4.8 Isi file obj.names

Dan yang terakhir adalah train.txt dan test.txt yang berfungsi untuk memberi tahu file mana yang akan digunakan untuk melakukan *training*, dan file mana yang akan digunakan untuk melakukan *testing.* contoh isi test.txt dapat dilihat pada gambar 4.9, dan contoh isi train.txt dapat dilihat pada gambar 4.10



Gambar 4.9 Isi file test.txt

Gambar 4.10 Isi file train.txt

Pembagian gambar pada train.txt, dan test.txt berasal dari program python Bernama process.py yang berfungsi untuk melakukan pembagian jumlah data *train*, dan *test* berdasarkan perintah yang ditulis.

Segmen Program 4.3 *Source code* untuk melakukan pembagian train.txt, dan test.txt

```
#!/usr/bin/env python
# coding: utf-8

# In[ ]:



import glob, os

# Current directory
current_dir = os.path.dirname(os.path.abspath(__file__))

print(current_dir)

current_dir = 'data/obj'

# Percentage of images to be used for the test set
percentage_test = 10;

# Create and/or truncate train.txt and test.txt
file_train = open('data/train.txt', 'w')
file_test = open('data/test.txt', 'w')

# Populate train.txt and test.txt
counter = 1
index_test = round(100 / percentage_test)
for pathAndFilename in glob.iglob(os.path.join(current_dir,
"*.jpg")):
      title, ext =
      os.path.splitext(os.path.basename(pathAndFilename))

      if counter == index_test:
```

```
            counter = 1
            file_test.write("data/obj" + "/" + title + '.jpg' + "\n")
        else
            file_train.write("data/obj" + "/" + title + '.jpg' + "\n")
            counter = counter + 1
```

Pada Segmen Program 4.3, *code* berfungsi untuk membaca dan mengambil seluruh nama file yang terdapat dalam suatu folder data, dan kemudian membagi data menjadi *training*, dan *testing*. Proses pembagian data wsendiri berdasarkan *code* diatas adalah 90% untuk *training*, dan 10% untuk *testing*.

4.3.1.2. Persiapan CFG Files

Sebelum melakukan *training YOLOv4 DarkNet*, perlu untuk melakukan pengaturan pada file yolov4-custom.cfg terlebih dahulu. Variabel yang harus diperhatikan dalam mengatur yolov4-custom.cfg adalah sebagai berikut:

Tabel 4.3 Konfigurasi YOLOv4

Nama	Deskripsi
batch	Jumlah batch tiap iterasi dalam satu epoch
subdivisions	Jumlah bagian batch dalam satu batch
max_batches	Jumlah maksimum iterasi dari *training*
steps	Penerapan skala pada *Checkpoint*
learning_rate	*Hyperparameter* yang berfungsi untuk mengontrol perubahan model dalam menanggapi kesalahan yang diperkirakan tiap kali bobot diperbarui
classes	Jumlah class yang di-*training*
filters	Jumlah *convolutional layers* yang akan digunakan

Pada Tabel 4.3, telah dijelaskan beberapa variabel yang harus diperhatikan. Untuk batch, digunakan 64 batch dengan 16 subdivisions berdasarkan *rule of thumb*. Lalu digunakan max_batches dengan rumus = (jumlah class * 2000). Akan tetapi, max_batches tidak dapat

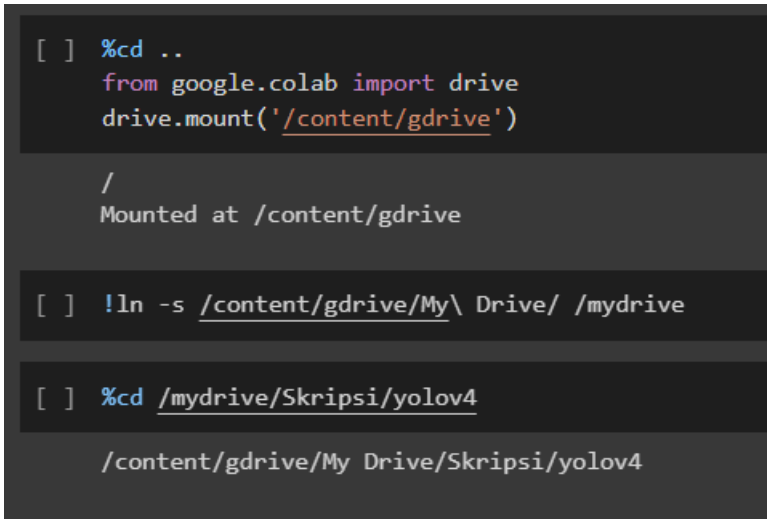
kurang dari 6000, dan tidak dapat kurang dari jumlah *training* gambar. Selanjutnya, steps memiliki rumus = (max_batches * 80%, max_bacthes * 90%) sehingga semakin besar max_batches, maka semakin besar juga steps nya. Lalu learning_rate yang digunakan untuk YOLOv4 adalah 0.001. Lalu yang terakhir, classes merupakan jumlah class yang akan dilakukan *training*, dan filters memiliki rumus= ((classes + 5) * 3). Contoh isi YOLOv4-custom.cfg *files* dapat dilihat pada Gambar 4.11

```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=16
width=320
height=320
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 158000
policy=steps
steps=126400,142200
scales=.1,.1

#cutmix=1
mosaic=1

#:104x104 54:52x52 85:26x26 104:13x13 for 416

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=mish

```

Gambar 4.11 YOLOv4-custom.cfg file

4.3.1.3. Training YOLOv4 model

Karena program dijalankan pada *Google Colab*, maka langkah pertama yang dilakukan adalah dengan melakukan *mount drive* pada *Google Colab*. *Command* untuk melakukan *mount* pada *Google Colab* dapat dilihat pada Gambar 4.12



Gambar 4.12 Mount Google Drive

Setelah Google Drive berhasil di-*mount*, maka akan dilakukan *git clone* dari website https://github.com/AlexeyAB/darknet. Ketika *clone* sudah selesai, maka perlu untuk mengubah *makefile* untuk mengaktifkan GPU dan cuDNN. Pada file Makefile, perlu untuk mengubah OPENCV, GPU, CUDNN, CUDNN_HALF, dan LIBSO menjadi 1 seperti yang tertera pada Segmen Program 4.4

Segmen Program 4.4 *Command* untuk mengubah *Makefile* pada python program

```
%cd darknet/
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
!sed -i 's/LIBSO=0/LIBSO=1/' Makefile
!make
```

Setelah diubah, dijalankan *command* !make untuk melakukan build pada DarkNet. Jika proses *build* DarkNet sudah selesai, maka langkah selanjutnya adalah melakukan pengunduhan *pretrained model* yang terdapat pada *website DarkNet* AlexeyAB . Apabila DarkNet telah di-*build*, dan *pretrained model* telah diunduh, maka dapat dilakukan *training.* Proses training dapat dijalankan dengan cara memasukkan *command* seperti yang tercantum pada Segmen Program 4.5.

Segmen Program 4.5 *Command training* YOLOv4

```
!./darknet detector train data/obj.data cfg/yolov4-custom.cfg
yolov4.conv.137 -dont_show -map
```

Seperti yang terlihat pada Segmen Program 4.5, yang dibutuhkan untuk melakukan *training* adalah:

- *Path Dataset*
- *Path* yolov4-custom.cfg
- *Path pretrained model* berupa *file* bernama yolov4.conv.137

4.3.2. Training YOLOv4-tiny

Pada YOLOv4-tiny, proses persiapan sebelum dilakukan *training,* sampai dengan proses *training* sama dengan YOLOv4. Akan tetapi, ada beberapa hal yang berbeda, seperti:

- Learning rate pada YOLOv4 adalah 0.001, sedangkan YOLOv4-tiny menggunakan 0.00261 sebagai learning rate
- Pada yolov4-custom.cfg, YOLOv4 memiliki 3 YOLO *layer*, dan 3 *convolutional layer*, sedangkan pada yolov4-tiny-custom.cfg hanya memiliki 2 *convolutional layer*, dan 2 YOLO *layer*
- *Pretrained model* yang digunakan oleh YOLO adalah yolov4.conv.137, sedangkan YOLOv4-tiny menggunakan yolov4-tiny.conv.29

Selain perbedaan yang telah disebutkan, proses persiapan data, dan pelaksanaan *training* YOLOv4-tiny sama dengan YOLOv4. Sehingga untuk melakukan training pada YOLOv4-tiny, digunakan *command* seperti pada Segmen Program 4.6

Segmen Program 4.6 *Command training* YOLOv4-tiny

```
!./darknet detector train data/obj.data cfg/yolov4-tiny-custom.cfg
yolov4-tiny.conv.29 -dont_show -map
```

Pada YOLOv4-tiny, yang dibutuhkan untuk melakukan *training* seperti yang terlihat pada Segmen Program 4.6 adalah

- *Path Dataset*
- *Path* yolov4-tiny-custom.cfg

- *Path pretrained model* berupa Yolov4-tiny.conv.29

4.3.3. Training AlexNet

Dataset yang digunakan dalam melatih AlexNet memiliki 69 class, dengan 100 data per class dan bukan hasil augmentasi gambar. Langkah awal yang harus dilakukan adalah dengan melakukan *import library* tensorflow, numpy, pathlib, dan datetime. Setelah itu akan dilakukan pembuatan nama kelas dengan cara melakukan *scan* dalam satu folder, dan mendapat setiap nama folder yang berada di dalamnya seperti pada Segmen Program 4.7

Segmen Program 4.7 Pembuatan CLASS_NAMES dari melakukan pengecekan dalam satu folder

```
# Dataset Directory

data_dir = pathlib.Path("E:/AlexNet_69_100/train_img")

image_count = len(list(data_dir.glob('*/*.png')))

print(image_count)

# classnames in the dataset specified

CLASS_NAMES = np.array([item.name for item in data_dir.glob('*')])

print(CLASS_NAMES)

# print length of class names

output_class_units = len(CLASS_NAMES)

print(output_class_units)
```

Setelah CLASS_NAMES didapatkan, maka Langkah selanjutnya adalah membuat model arsitektur dari AlexNet. Proses pembuatan model dapat dilihat pada Segmen Program 4.8

Segmen Program 4.8 Membuat model arsitektur AlexNet

```
model = tf.keras.models.Sequential([
```

```
      # 1st conv
   tf.keras.layers.Conv2D(96, (11,11),strides=(4,4),
   activation='relu', input_shape=(227, 227, 3)),
   tf.keras.layers.BatchNormalization(),
   tf.keras.layers.MaxPooling2D(2, strides=(2,2)),
      # 2nd conv
   tf.keras.layers.Conv2D(256, (11,11),strides=(1,1),
   activation='relu',padding="same"),
   tf.keras.layers.BatchNormalization(),
      # 3rd conv
   tf.keras.layers.Conv2D(384, (3,3),strides=(1,1),
   activation='relu',padding="same"),
   tf.keras.layers.BatchNormalization(),
      # 4th conv
   tf.keras.layers.Conv2D(384, (3,3),strides=(1,1),
   activation='relu',padding="same"),
   tf.keras.layers.BatchNormalization(),
      # 5th Conv
   tf.keras.layers.Conv2D(256, (3, 3), strides=(1, 1),
   activation='relu',padding="same"),
   tf.keras.layers.BatchNormalization(),
   tf.keras.layers.MaxPooling2D(2, strides=(2, 2)),
   # To Flatten layer
   tf.keras.layers.Flatten(),
   # To FC layer 1
   tf.keras.layers.Dense(1024, activation='relu'),
   tf.keras.layers.Dropout(0.5),
   #To FC layer 2
   tf.keras.layers.Dense(1024, activation='relu'),
   tf.keras.layers.Dropout(0.5),
   tf.keras.layers.Dense(output_class_units, activation='softmax')
])
```

Pada Segmen Program 4.8, dimensi gambar yang dimasukkan adalah 227 * 227 dikarenakan untuk mendapatkan size sebesar 55 pada *convolution layer* pertama dengan 11*11 *kernels*, 4 *stride,* dan 2 *padding* dibutuhkan dimensi 227. Rumus untuk mencari size adalah sebagai berikut:

$$size = \frac{(dimension+2*padding-kernel)}{stride} + 1 \quad (4.1)$$

Setelah model arsitektur dari AlexNet telah selesai dibuat, selanjutnya adalah menyiapkan data, dan melakukan *preprocessing* untuk dilakukan *training.* Proses penyiapan data seperti menentukan batch_size, *height, width, steps per epoch,* melakukan *resize,* dan *preprocessing* gambar dapat dilihat pada Segmen Program 4.9

Segmen Program 4.9 Penyiapan data , dan *preprocessing* sebelum dilakukan *training*

```
# Shape of inputs to NN Model
BATCH_SIZE = 32             # Can be of size 2^n, but not restricted
to. for the better utilization of memory
IMG_HEIGHT = 227            # input Shape required by the model
IMG_WIDTH = 227             # input Shape required by the model
STEPS_PER_EPOCH = np.ceil(image_count/BATCH_SIZE)

# Rescalingthe pixel values from 0~255 to 0~1 For RGB Channels of
the image.
image_generator =
tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
# training_data for model training
train_data_gen =
image_generator.flow_from_directory(directory=str(data_dir),
batch_size=BATCH_SIZE,
                        shuffle=True,
target_size=(IMG_HEIGHT, IMG_WIDTH), #Resizing the raw dataset
                        classes = list(CLASS_NAMES))
```

Dapat dilihat pada Segmen Program 4.9 dibuat *generator* untuk melakukan *training* data dengan batch size, *width & height* yang telah ditentukan sebelumnya, dan mengambil CLASS_NAME sebagai classes. Setelah dilakukan penyiapan data, dan *preprocessing* data, lalu dibutuhkan untuk melakukan fungsi *compile* pada model agar dapat menspesifikasikan *optimizer,* dan menampilkan *loss function* & *metrics* seperti yang tertera pada Segmen Program 4.10

Segmen Program 4.10 Menspesifikasikan *optimizer*, menampilkan *loss function* & *metrics*, dan menampilkan *summary* dari model arsitektur

```
# Specifying the optimizer, Loss function for optimization & Metrics
to be displayed
model.compile(optimizer='sgd', loss="categorical_crossentropy",
metrics=['accuracy'],steps_per_execution=100)

# Summarizing the model architecture and printing it out
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 55, 55, 96)        34944

 batch_normalization (BatchN  (None, 55, 55, 96)       384
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 27, 27, 96)       0
 )

 conv2d_1 (Conv2D)           (None, 27, 27, 256)       2973952

 batch_normalization_1 (Batc  (None, 27, 27, 256)      1024
 hNormalization)

 conv2d_2 (Conv2D)           (None, 27, 27, 384)       885120

 batch_normalization_2 (Batc  (None, 27, 27, 384)      1536
 hNormalization)

 conv2d_3 (Conv2D)           (None, 27, 27, 384)       1327488

 batch_normalization_3 (Batc  (None, 27, 27, 384)      1536
 hNormalization)

 conv2d_4 (Conv2D)           (None, 27, 27, 256)       884992

 batch_normalization_4 (Batc  (None, 27, 27, 256)      1024
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 13, 13, 256)      0
 2D)

 flatten (Flatten)           (None, 43264)             0

 dense (Dense)               (None, 1024)              44303360

 dense_1 (Dense)             (None, 1024)              1049600

 dense_2 (Dense)             (None, 69)                70725

=================================================================
Total params: 51,535,685
Trainable params: 51,532,933
Non-trainable params: 2,752
_________________________________________________________________
```

Gambar 4.13 *Summary* model arsitektur AlexNet

Setelah dilakukan pengecekan terhadap *summary* seperti yang terlihat pada Gambar 4.13, selanjutnya adalah membuat fungsi *Callback* yang bertujuan untuk menghentikan *training* apabila akurasi telah mencapai 100%. Folder logs juga disimpan agar hasil *training* dapat dilihat pada Tensorboard. *Command* untuk fungsi *Callback* dapat dilihat pada Segmen Program 4.11

Segmen Program 4.11 Membuat fungsi Callback, dan menyimpan logs folder

```
# callbacks at training
class myCallback(tf.keras.callbacks.Callback):
      def on_epoch_end(self,epoch,logs={}):
```

```
        if (logs.get("accuracy")==1.00 and logs.get("loss")<0.03):
            print("\nReached 100% accuracy so stopping training")
            self.model.stop_training =True
callbacks = myCallback()

# TensorBoard.dev Visuals
log_dir="logs\\fit\\" + datetime.datetime.now().strftime("%Y%m%d-
%H%M%S")
tensorboard_callback =
tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
```

Setelah semua hal tersebut telah dilakukan, maka *training* dapat dilakukan. Perintah untuk melakukan *training* dapat dilihat pada Segmen Program 4.12

Segmen Program 4.12 *Training* AlexNet

```
# Training the Model
history = model.fit(
      train_data_gen,
      steps_per_epoch=STEPS_PER_EPOCH,
      epochs=50,
      callbacks=[tensorboard_callback,callbacks])

# Saving the model
model.save('AlexNet_saved_model/')
```

4.4. Implementasi Testing

Pada subbab ini akan dijelaskan mengenai implementasi testing untuk YOLOv4, YOLOv4-tiny, dan AlexNet beserta data yang digunakan dalam proses testing yang dilakukan

4.4.1. Testing YOLOv4

Pada segmen ini akan dijelaskan mengenai program yang akan dijalankan agar proses testing data dapat berjalan

Sebelum dilakukan *testing*, ada beberapa hal yang perlu dipersiapkan terlebih dahulu seperti yolov4-custom.cfg. Contoh untuk persiapan *batch*, dan *subdivisions* yolov4 dapat dilihat pada Segmen Program 4.13

Segmen Program 4.13 Mengubah batch dan subdivisions yolov4-custom.cfg untuk testing

```
%cd cfg
!sed -i 's/batch=64/batch=1/' yolov4-custom.cfg
!sed -i 's/subdivisions=16/subdivisions=1/' yolov4-custom.cfg
%cd ..
```

Setelah selesai mengubah yolov4-custom.cfg, maka proses testing image dapat dilakukan. Perintah untuk melakukan testing gambar dapat dilihat pada Segmen Program 4.14

Segmen Program 4.14 Testing image YOLOv4

```
!./darknet detector map data/obj.data cfg/yolov4-custom.cfg
/mydrive/yolov4/training/yolov4-custom_best.weights -points 0
```

4.4.2. Testing YOLOv4-tiny

Hampir tidak ada perbedaan pada cara untuk melakukan testing pada YOLOv4-tiny dan YOLOv4 kecuali nama file yaitu yolov4-tiny-custom.cfg, dan modelnya yaitu yolov4-tiny-custom_best.weights . Pertama, sama seperti YOLOv4, akan dilakukan pengubahan batch, dan subdivision menjadi 1. Kemudian setelah diubah, proses testing YOLOv4-tiny dapat berjalan dengan perintah yang dapat dilihat pada Segmen Program 4.15

Segmen Program 4.15 Testing image YOLOv4-tiny

```
!./darknet detector map data/obj.data cfg/yolov4-tiny-custom.cfg
/mydrive/yolov4/training/yolov4-tiny-custom_best.weights -points 0
```

4.4.3. Testing AlexNet

Untuk melakukan testing pada AlexNet, pertama perlu untuk melakukan pengecekan data *test image* apakah jumlah gambar, dan nama *class* sudah ada dan sesuai. Setelah itu, dilakukan *preprocessing* kembali dan mengatur *batch size testing* menjadi 1, dan dimensi gambar menjadi 227 * 227. Setelah dilakukan *preprocessing*, maka model yang telah disimpan

dapat di-*load* dan dilakukan *testing*. *Command* untuk *Testing model* dapat dilihat pada Segmen Program 4.16

Segmen Program 4.16 Testing model arsitektur AlexNet

```
# import necessary package
import tensorflow as tf
import numpy as np
import pathlib
import datetime
# Raw Dataset Directory
data_dir = pathlib.Path("E:/AlexNet_69_100/test_img")
image_count = len(list(data_dir.glob('*/*.png')))
# print total no of images for all classes
print(image_count)
# classnames in the dataset specified
CLASS_NAMES = np.array([item.name for item in data_dir.glob('*')])
# print list of all classes
print(CLASS_NAMES)
# print length of class names
output_class_units = len(CLASS_NAMES)
print(output_class_units)
#preprocess the data
BATCH_SIZE = 1             # Can be of size 2^n, but not restricted
to. for the better utilization of memory
IMG_HEIGHT = 227            # input Shape required by the model
IMG_WIDTH = 227             # input Shape required by the model

image_generator =
tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
test_data_gen =
image_generator.flow_from_directory(directory=str(data_dir),
                        shuffle=True,
target_size=(IMG_HEIGHT, IMG_WIDTH), #Resizing the raw dataset
                        classes = list(CLASS_NAMES))
#Loading the saved model
new_model = tf.keras.models.load_model("AlexNet_saved_model.h5")
new_model.summary()
loss, acc = new_model.evaluate(test_data_gen)
```

```
print("accuracy:{:.2f}%".format(acc*100))
```

4.5. Implementasi Aplikasi

Pada segmen ini, akan dijelaskan mengenai proses pembuatan GUI (*Graphical User Interface*) untuk menampilkan gambar yang ingin diprediksi. Model / Weight yang diambil adalah weight dengan 69 class, dan 100 data.

Segmen Program 4.17 Fungsi untuk menampilkan data hasil prediksi

```
def imShow(path):
   import cv2
   import matplotlib.pyplot as plt
   %matplotlib inline
   image = cv2.imread(path)
   height, width = image.shape[:2]
   resized_image = cv2.resize(image,(3*width, 3*height),
   interpolation = cv2.INTER_CUBIC)
   fig = plt.gcf()
   fig.set_size_inches(18, 10)
   plt.axis('off')
   plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
   plt.show()
```

Sebelum GUI dibuat, harus dibuat fungsi imShow terlebih dahulu agar gambar dapat ditampilkan pada GUI seperti pada Segmen Program 4.17 Setelah fungsi selesai dibuat, maka harus dipastikan terlebih dahulu apakah *darknet* sudah di-*build* menggunakan *CMake* atau belum. Jika sudah, maka GUI dapat dibuat menggunakan program python. *Command* untuk pembuatan GUI dan implementasi YOLO ke dalam GUI dapat dilihat pada Segmen Program 4.18

Segmen Program 4.18 Pembuatan GUI dan implementasi YOLO

```
# image_viewer.py
import io
import os
import sys
import PySimpleGUI as sg
from PIL import Image
file_types = [("JPEG (*.jpg)", "*.jpg"),
```

```
                ("All files (*.*)", "*.*")]

l_column=[
      [
         sg.Text("Image File"),
         sg.Input(size=(25, 1), key="-FILE-"),
         sg.FileBrowse(file_types=file_types),
         sg.Button("Load Image"),
      ],
      [
         sg.Image(key="-IMAGE-"),
      ],
      [
         sg.Button("Identify Image"),
      ]
]

r_column=[
      [sg.Text("Identified Image will show here")],
      [sg.Image(key="-ID-")]
]



layout = [
      [
         sg.Column(l_column),
         sg.VSeperator(),
         sg.Column(r_column),
      ]
]

window = sg.Window("Javanese Script Identifier", layout)

while True:
      event, values = window.read()
      if event == "Exit" or event == sg.WIN_CLOSED:
         break
      if event == "Load Image":
```

```
            filename = values["-FILE-"]
            if os.path.exists(filename):
               image = Image.open(values["-FILE-"])
               image.thumbnail((400, 400))
               bio = io.BytesIO()
               image.save(bio, format="PNG")
               window["-IMAGE-"].update(data=bio.getvalue())
         if event == "Identify Image":
            name = values["-FILE-"]
            print(values["-FILE-"])
            if os.path.exists(name):
               !darknet.exe detector test data/obj.data cfg/yolov4-
               custom.cfg yolov4-custom.weights $name -thresh 0.1 -
               dont_show
               imShow('predictions.jpg')
               im = Image.open('predictions.jpg')
               bio1 = io.BytesIO()
               im.save(bio1, format="PNG")
               window["-ID-"].update(data=bio1.getvalue())



window.close()
```

Dibutuhkan library PySimpleGUI untuk membuat GUI pada *Python*. Implementasi *YOLOv4* dilakukan ketika tombol *Identify Image* ditekan. Proses yang terjadi yaitu *darknet.exe* akan dijalankan melalui terminal *command prompt* (cmd) untuk dilakukan prediksi pada gambar yang dimasukkan menggunakan variabel $name dengan memanggil *weight* yang telah di-*train*, yolov4-custom.cfg, dan obj.data.