International Refereed Journal of Engineering and Science (IRJES)
ISSN (Online) 2319-183X, (Print) 2319-1821
Volume 13, Issue 5 (Sep.-Oct. 2024), PP. 45-62

The Performance of Color and Grayscale Image Input in Convolutional Neural Network based Head Detection

Panca Mudjirahardjo*, Aqil Gama Rahmansyah*, Alya Shafa Dianti**
** (Department of Electrical Engineering, Faculty of Engineering, Universitas Brawijaya, Malang-Indonesia)*
*** (Department of Statistics, Faculty of Mathematics and Natural Sciences, Universitas Brawijaya, Malang-Indonesia)*
Corresponding Author: panca@ub.ac.id

Abstract: *In this paper we study the performance of color and grayscale image input for CNN architecture for head detection. We study five architecture models, in varies of feature extraction stages and the number of neural network layers. In training phase, all models have good validation accuracy, i.e. above 90%. However, in implementation for head detection, they have low precision value, due to the large number of false positive. The high performance of head detection is achieved by grayscale processing and image size of 30×20 pixels for training datasets.*

Keywords: *Head detection; Convolutional Neural Network; optimizer; performance evaluation.*

Date of Submission: 09-08-2024 Date of acceptance: 25-09-2024

I. INTRODUCTION

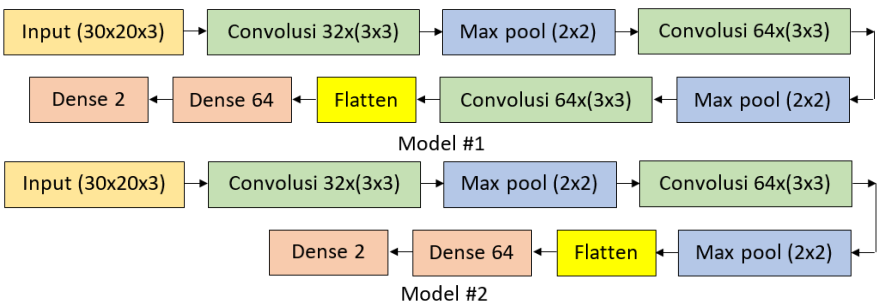
Head detection in computer vision is an important task for automatic systems, such as smart room monitoring, surveillance system, security system, and so on. Head detection searches for a head or heads in a given image, then localizes them for further processing. This task is more difficult and complicated than head classification [1][2]. Head classification is simply to verify whether an image belongs to the head class or not. Head detection becomes difficult when the head pose varies greatly and the background also varies. The head is not only facing forward, but also backward, sideways, looking up and down. The background is also not static, but also textured.

There are many object classification method implemented in automation system. However, nowadays, CNN become a popular architecture in object classification. This due to the feature extraction and classifier task to be done in sequential process. Convolutional stage is a feature extraction, followed by a neural network as an object classifier.

In recent years, Convolutional Neural Networks (CNNs) have revolutionized the field of artificial intelligence, particularly in the areas of image and video recognition, natural language processing, and beyond. A CNN is a type of deep learning model specifically designed to process and analyze visual data. Unlike traditional neural networks, which work with vectorized inputs, CNNs are adept at recognizing patterns and spatial hierarchies in grid-like data, such as images [3][4][5][6].

II. STUDY METHOD

In this section we introduce and explain the method of our study. The feature extraction is performed by convolution and max pooling stages, and the classifier stage is performed by flatten and dense stages. Input images are size of 30×20 pixels and 20×20 pixels, color and grayscale format. We study some CNN architectures as depicted in Fig. 1. We use INRIA datasets for training data [7], as shown in Fig. 2.



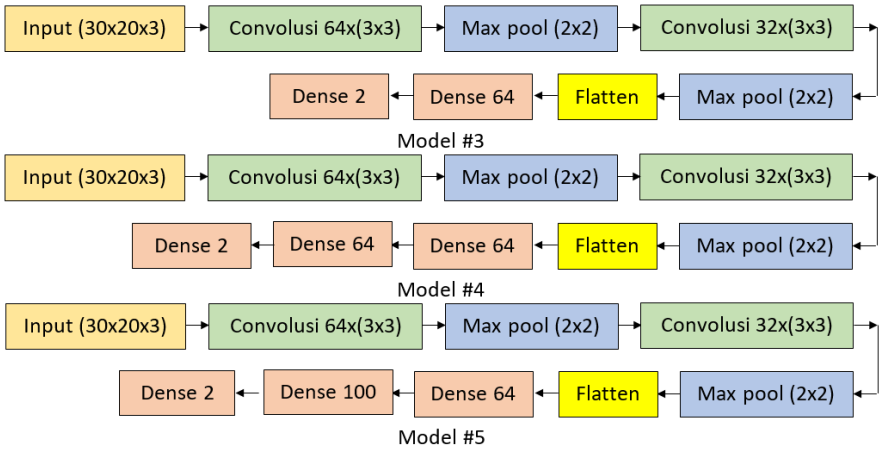


Figure 1. CNN architectures in our study

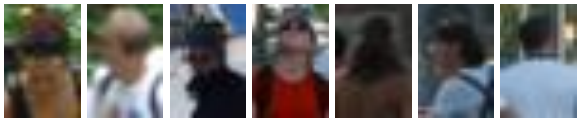


Figure 2. Some of head in INRIA datasets [7]

2.1 Training phase for image size 30×20 pixels, color format.

```
print('CNN training, 30x20 COLOR ..\n'*5)
print("ARZETI_Getsuyoubi,12.08.2024; 03:43")
print("Panca" +
      " Mudjirahardjo")
print("")
print("================================================================================================")

print("")
print("")

modelKE = input('What model (1,2,3) : ')

print('')
optim = input('Optimizer: (1)SGD, (2)ADAM, (3)RMSprop, (4)Adagrad, (5)AdamW, (6)Adadelta, (7)Adamax, (8)Nadam : ')
if optim=='1':
    optim='SGD'
    print('-- optimizer: SGD')
elif optim=='2':
    optim='ADAM'
    print('-- optimizer: ADAM')
elif optim=='3':
    optim='RMSprop'
    print('-- optimizer: RMSprop')
elif optim=='4':
    optim='Adagrad'
    print('-- optimizer: Adagrad')
elif optim=='5':
    optim='AdamW'
    print('-- optimizer: AdamW')
elif optim=='6':
    optim='Adadelta'
    print('-- optimizer: Adadelta')
elif optim=='7':
    optim='Adamax'
```

```
    print('-- optimizer: Adamax')
elif optim=='8':
    optim='Nadam'
    print('-- optimizer: Nadam')

print('')
ep = input('Jumlah epoch (1 epoch 42 sec !): ')
ep = int(ep)

# ------------------------------------------------------------------------------

import numpy as np
import os
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'

import tensorflow as tf
from tensorflow.keras import layers, models

# ------------------------------------------------------------------------------

def model_1():
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(30, 20, 3)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(num_classes))

    print('')
    model.summary()
    return model

def model_2():
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(30, 20, 3)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(num_classes))

    print('')
    model.summary()
    return model

def model_3():
    model = models.Sequential()
    model.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(30, 20, 3)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(32, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(num_classes))

    print('')
```

```
    model.summary()
    return model

# ------------------------------------------------------------------------------------

data_dir = "D:\Datasets"

train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.25,
  subset="training",
  seed=123,
  image_size=(30, 20),
  batch_size=20)

val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.25,
  subset="validation",
  seed=123,
  image_size=(30, 20),
  batch_size=20)

class_names = train_ds.class_names
print(class_names)

import matplotlib.pyplot as plt

for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break

AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

num_classes = 2

# ------------------------------------------------------------------------------------

print("")
print("")
if modelKE == '1':
    model = model_1()
    print("")
    print('---- model ke 1 (satu) ---')
elif modelKE == '2':
    model = model_2()
    print("")
    print('---- model ke 2 (dua) ---')
elif modelKE == '3':
    model = model_3()
    print("")
    print('---- model ke 3 (tiga) ---')
print('')
print('---- Optimizer: ' +optim)
print('---- training dimulai --- ')
print('')
```

```
model.compile(
  optimizer=optim,
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])

history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=ep
)

plt.figure('Model: ' +modelKE +' , optimizer: ' +optim)
plt.plot(history.history['accuracy'], label='train_accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
plt.show()

print('')
print(' ----- model evaluate ---- ')
test_loss, test_acc = model.evaluate(val_ds)        # , verbose=1

model.save('D:\Program\python 3.11.5\Training model\my_model.keras')
```

2.2 Training phase for image size 30×20 pixels, grayscale format.

```
print('CNN training 30x20 Grayscale ..\n'*5)
print("ARZETI_Doyoubi,24.08.2024; 08:20")
print("Panca" +
      " Mudjirahardjo")
print("")
print("================================================================")

print("")

modelKE = input('What model (1,2,3,4,5) : ')

print('')
optim = input('Optimizer: (1)SGD, (2)ADAM, (3)RMSprop, (4)Adagrad, (5)AdamW, (6)Adadelta, (7)Adamax, (8)Nadam : ')
if optim=='1':
    optim='SGD'
    print('-- optimizer: SGD')
elif optim=='2':
    optim='ADAM'
    print('-- optimizer: ADAM')
elif optim=='3':
    optim='RMSprop'
    print('-- optimizer: RMSprop')
elif optim=='4':
    optim='Adagrad'
    print('-- optimizer: Adagrad')
elif optim=='5':
    optim='AdamW'
    print('-- optimizer: AdamW')
elif optim=='6':
```

```
    optim='Adadelta'
    print('-- optimizer: Adadelta')
elif optim=='7':
    optim='Adamax'
    print('-- optimizer: Adamax')
elif optim=='8':
    optim='Nadam'
    print('-- optimizer: Nadam')

print('')
ep = input('Jumlah epoch : ')
ep = int(ep)

# ------------------------------------------------------------------------

import numpy as np
import os
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'

import tensorflow as tf
from tensorflow.keras import layers, models

# ------------------------------------------------------------------------

def model_1():
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(30, 20, 1)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(num_classes))

    print('')
    model.summary()
    return model

def model_2():
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(30, 20, 1)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(num_classes))

    print('')
    model.summary()
    return model

def model_3():
    model = models.Sequential()
    model.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(30, 20, 1)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(32, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
```

```
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(num_classes))

    print('')
    model.summary()
    return model

def model_4():
    model = models.Sequential()
    model.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(30, 20, 1)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(32, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(num_classes))

    print('')
    model.summary()
    return model

def model_5():
    model = models.Sequential()
    model.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(30, 20, 1)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(32, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(100, activation='relu'))
    model.add(layers.Dense(num_classes))

    print('')
    model.summary()
    return model

# ------------------------------------------------------------------------

data_dir = "D:\Datasets"

train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.25,
  subset="training",
  seed=123,
  color_mode="grayscale",
  image_size=(30, 20),
  batch_size=20)

val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.25,
  subset="validation",
  seed=123,
  color_mode="grayscale",
  image_size=(30, 20),
  batch_size=20)

class_names = train_ds.class_names
```

```
print(class_names)

import matplotlib.pyplot as plt

for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break

AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

num_classes = 2

# ------------------------------------------------------------------------

print('')
print('---- Optimizer: ' +optim)
print('---- training dimulai --- ')
print('')

model.compile(
  optimizer=optim,
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])

history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=ep
)

plt.figure('Model: ' +modelKE +' , optimizer: ' +optim)
plt.plot(history.history['accuracy'], label='train_accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
plt.show()

print('')
print(' ----- model evaluate ---- ')
test_loss, test_acc = model.evaluate(val_ds)

model.save('D:\Program\python 3.11.5\Training model\my_model.keras')
```

2.3 Head Detection for image size 30×20 pixels, color format.

```
print('CNN head detection, 30x20 COLOR ..\n'*5)
print("ARZETI_Kinyoubi,13.09.2024; 02:31")
print("Panca" +
     " Mudjirahardjo")
print("")
```

```
print("================================================================================================")

print('')
print('-- import library ---')
print('')

import numpy as np
import os
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'

#import PIL
#from PIL import Image
import cv2 as cv
import tensorflow as tf
from tensorflow.keras import models
from keras.preprocessing import image

# ------------------------------------------------------------------------------

people = cv.imread("D:\Program\output image\people.jpg")
cv.imshow('people',people)
cv.waitKey(0)

print('')
print('Model: ')
print('  1. model_1_30x20_adam_50_t1500v500_png_jpg.keras')
print('  2. model_2_30x20_adam_50_t1500v500_png_jpg.keras')
print('  3. model_3_30x20_adam_50_t1500v500_png_jpg.keras')
print('')
print('  4. model_1_30x20_rmsprop_50_t1500v500_png_jpg.keras')
print('  5. model_2_30x20_rmsprop_50_t1500v500_png_jpg.keras')
print('  6. model_3_30x20_rmsprop_50_t1500v500_png_jpg.keras')
print('')
modelKE = input('    model ke: ')

people = ["person_1.jpg","person_2.jpg","person_3.jpg","person_4.jpg","person_5.jpg","person_6.jpg","person_7.jpg",
          "person_8.jpg","person_9.jpg","person_10.jpg","person_11.jpg","person_12.jpg","person_13.jpg","person_14.jpg",
          "person_15.jpg","person_16.jpg","person_17.jpg","person_18.jpg","person_19.jpg","person_20.jpg"]

org = []
orgKe = 0
for a in people:
    path = 'D:/Program/output image/' +a
    oriIMG = cv.imread(path)
    print(a)

    newIMG = oriIMG.copy()
    grayIMG = cv.cvtColor(oriIMG, cv.COLOR_BGR2GRAY )

    h,w,c = oriIMG.shape
    print(oriIMG.shape)

    print('')
    print('-- loading model ---')

    if modelKE=='1':
        model = models.load_model('D:\Program\python 3.11.5\Training
model\model_1_30x20_adam_50_t1500v500_png_jpg.keras')
        print('-- model: model_1_30x20_adam_50_t1500v500_png_jpg.keras')
    elif modelKE=='2':
```

```
        model = models.load_model('D:\Program\python 3.11.5\Training
model\model_2_30x20_adam_50_t1500v500_png_jpg.keras')
        print('-- model: model_2_30x20_adam_50_t1500v500_png_jpg.keras')
    elif modelKE=='3':
        model = models.load_model('D:\Program\python 3.11.5\Training
model\model_3_30x20_adam_50_t1500v500_png_jpg.keras')
        print('-- model: model_3_30x20_adam_50_t1500v500_png_jpg.keras')
    elif modelKE=='4':
        model = models.load_model('D:\Program\python 3.11.5\Training
model\model_1_30x20_rmsprop_50_t1500v500_png_jpg.keras')
        print('-- model: model_1_30x20_rmsprop_50_t1500v500_png_jpg.keras')
    elif modelKE=='5':
        model = models.load_model('D:\Program\python 3.11.5\Training
model\model_2_30x20_rmsprop_50_t1500v500_png_jpg.keras')
        print('-- model: model_2_30x20_rmsprop_50_t1500v500_png_jpg.keras')
    elif modelKE=='6':
        model = models.load_model('D:\Program\python 3.11.5\Training
model\model_3_30x20_rmsprop_50_t1500v500_png_jpg.keras')
        print('-- model: model_3_30x20_rmsprop_50_t1500v500_png_jpg.keras')

    # ------------------------------------------------------------------------------

    print('')

    # ------------------------------------------------------------------------------

    i=0
    pos = 0
    for r in range(0,h-30,15):
        for c in range(0,w-20,10):
            i=i+1
            print('blok ke: ',i)

            cropped_image = oriIMG[r:r+30,c:c+20]
            test_image = image.img_to_array(cropped_image)
            test_image = np.expand_dims(test_image, axis = 0)
            test_image = np.reshape(test_image,(30,20,3))
            test_image = np.expand_dims(test_image, axis=0)
            result_prob = model.predict(test_image)

            result_label = tf.argmax(result_prob, axis=-1).numpy()[0]
            #cv.rectangle(newIMG, pt1=(c,r), pt2=(c+20,r+30), color=(0,255,255), thickness=1)

            if result_label == 1:
                cv.rectangle(newIMG, pt1=(c,r), pt2=(c+20,r+30), color=(0,255,0), thickness=1)
                pos = pos + 1

    org.append(pos)

    orgKe = orgKe + 1

    print('')
    print('')

    cv.imwrite('D:/Program/output image/out_1_' +a,newIMG)

    print('')
    print('')
    print('--- model ke: ',modelKE)
    print('--- '+a +' completed! ')
```

```
    print('')
    print('')

print(org)
```

2.4 Head Detection for image size 30×20 pixels, grayscale format.

```
print('CNN head detect, 30x20 GRAYSCALE, ..\n'*5)
print("ARZETI_Doyoubi,24.08.2024; 09:46")
print("Panca" +
      " Mudjirahardjo")
print("")
print("================================================================================================")

print('')
print('-- import library ---')
print('')

import numpy as np
import os
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'

#import PIL
#from PIL import Image
import cv2 as cv
import tensorflow as tf
from tensorflow.keras import models
from keras.preprocessing import image

# ------------------------------------------------------------------------------

people = cv.imread("D:\Program\output image\people.jpg")
cv.imshow('people',people)
cv.waitKey(0)

print('')
print('Model: ')
print('  1. model_1_30x20_adam_50_t1500v500_gray.keras')
print('  2. model_2_30x20_adam_50_t1500v500_gray.keras')
print('  3. model_3_30x20_adam_50_t1500v500_gray.keras')
print('')
print('  4. model_1_30x20_RMSprop_50_t1500v500_gray.keras')
print('  5. model_2_30x20_RMSprop_50_t1500v500_gray.keras')
print('  6. model_3_30x20_RMSprop_50_t1500v500_gray.keras')
print('')
print('  7. model_4_30x20_adam_50_t1500v500_gray.keras')
print('  8. model_4_30x20_RMSprop_50_t1500v500_gray.keras')
print('')
print('  9. model_5_30x20_adam_50_t1500v500_gray.keras')
print('  10. model_5_30x20_RMSprop_50_t1500v500_gray.keras')
print('')
modelKE = input('   model ke: ')

people = ["person_1.jpg","person_2.jpg","person_3.jpg","person_4.jpg","person_5.jpg","person_6.jpg","person_7.jpg",
        "person_8.jpg","person_9.jpg","person_10.jpg","person_11.jpg","person_12.jpg","person_13.jpg","person_14.jpg",
        "person_15.jpg","person_16.jpg","person_17.jpg","person_18.jpg","person_19.jpg","person_20.jpg"]

org = []
```

```
orgKe = 0
for a in people:
    path = 'D:/Program/output image/' +a
    oriIMG = cv.imread(path)
    print(a)

    newIMG = oriIMG.copy()
    grayIMG = cv.cvtColor(oriIMG, cv.COLOR_BGR2GRAY )

    h,w,c = oriIMG.shape
    print(oriIMG.shape)

    print('')
    print('-- loading model ---')

    if modelKE=='1':
        model = models.load_model('D:\Program\python 3.11.5\Training
model\model_1_30x20_adam_50_t1500v500_gray.keras')
        print('-- model: model_1_30x20_adam_50_t1500v500_gray.keras')
    elif modelKE=='2':
        model = models.load_model('D:\Program\python 3.11.5\Training
model\model_2_30x20_adam_50_t1500v500_gray.keras')
        print('-- model: model_2_30x20_adam_50_t1500v500_gray.keras')
    elif modelKE=='3':
        model = models.load_model('D:\Program\python 3.11.5\Training
model\model_3_30x20_adam_50_t1500v500_gray.keras')
        print('-- model: model_3_30x20_adam_50_t1500v500_gray.keras')
    elif modelKE=='4':
        model = models.load_model('D:\Program\python 3.11.5\Training
model\model_1_30x20_RMSprop_50_t1500v500_gray.keras')
        print('-- model: model_1_30x20_RMSprop_50_t1500v500_gray.keras')
    elif modelKE=='5':
        model = models.load_model('D:\Program\python 3.11.5\Training
model\model_2_30x20_RMSprop_50_t1500v500_gray.keras')
        print('-- model: model_2_30x20_RMSprop_50_t1500v500_gray.keras')
    elif modelKE=='6':
        model = models.load_model('D:\Program\python 3.11.5\Training
model\model_3_30x20_RMSprop_50_t1500v500_gray.keras')
        print('-- model: model_3_30x20_RMSprop_50_t1500v500_gray.keras')
    elif modelKE=='7':
        model = models.load_model('D:\Program\python 3.11.5\Training
model\model_4_30x20_adam_50_t1500v500_gray.keras')
        print('-- model: model_4_30x20_adam_50_t1500v500_gray.keras')
    elif modelKE=='8':
        model = models.load_model('D:\Program\python 3.11.5\Training
model\model_4_30x20_rmsprop_50_t1500v500_gray.keras')
        print('-- model: model_4_30x20_rmsprop_50_t1500v500_gray.keras')
    elif modelKE=='9':
        model = models.load_model('D:\Program\python 3.11.5\Training
model\model_5_30x20_adam_50_t1500v500_gray.keras')
        print('-- model: model_5_30x20_adam_50_t1500v500_gray.keras')
    elif modelKE=='10':
        model = models.load_model('D:\Program\python 3.11.5\Training
model\model_5_30x20_RMSprop_50_t1500v500_gray.keras')
        print('-- model: model_5_30x20_RMSprop_50_t1500v500_gray.keras')

    # ------------------------------------------------------------------------------

    print('')
```

```
    # ------------------------------------------------------------------------------

    i=0
    pos = 0
    for r in range(0,h-30,15):
        for c in range(0,w-20,10):
            i=i+1
            print('blok ke: ',i)

            cropped_image = grayIMG[r:r+30,c:c+20]
            test_image = image.img_to_array(cropped_image)
            test_image = np.expand_dims(test_image, axis = 0)
            test_image = np.reshape(test_image,(30,20))
            test_image = np.expand_dims(test_image, axis=0)
            result_prob = model.predict(test_image)

            result_label = tf.argmax(result_prob, axis=-1).numpy()[0]
            #cv.rectangle(newIMG, pt1=(c,r), pt2=(c+20,r+30), color=(0,255,255), thickness=1)

            if result_label == 1:
                cv.rectangle(newIMG, pt1=(c,r), pt2=(c+20,r+30), color=(0,255,0), thickness=1)
                pos = pos + 1

    org.append(pos)

    orgKe = orgKe + 1

    print('')
    print('')

    cv.imwrite('D:/Program/output image/out_1_' +a,newIMG)

    print('')
    print('')
    print('--- model ke: ',modelKE)
    print('--- '+a +' completed! ')
    print('')
    print('')

print(org)
```

III. THE EXPERIMENTAL RESULT

This section we explain the experimental result. We perform the experiment to classify two classes, head and no-head class. As images in Figure 2 show some of head data for head class and another images are no-head class. We have 4000 image files belonging to 2 classes, using 3000 files for training and 1000 files for validation.

We put the images in directory image:

D:\Dataset\
- Head_OK\
  - head_ok(1).png
  - head_ok(2).png
  - head_ok(3).png
  - ---
  - ---
- Head_NG\
  - head_ng(1).png
  - head_ng(2).png
  - head_ng(3).png

We use images in Figure 3 to evaluate the CNN architecture in Figure 1.

Figure 3. Some of the scenes used for this experiment [7]

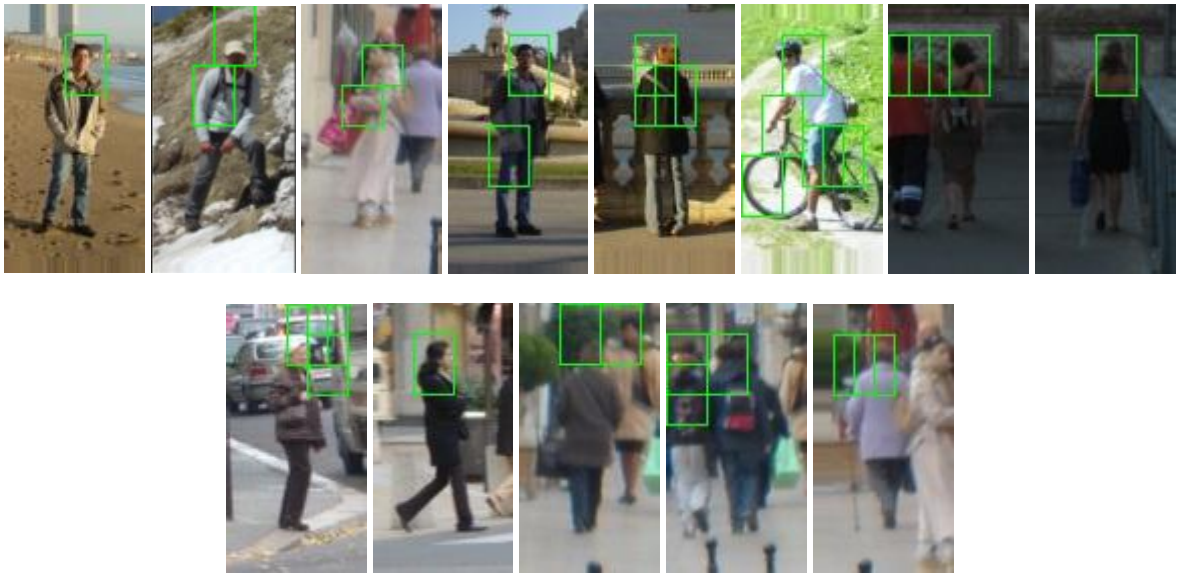


Figure 4. Some of expected and detected result

Model summary of model-1 – model-5, as shown in Figure 5-9 respectively.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 18, 32)	896
max_pooling2d (MaxPooling2D)	(None, 14, 9, 32)	0
conv2d_1 (Conv2D)	(None, 12, 7, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 3, 64)	0
conv2d_2 (Conv2D)	(None, 4, 1, 64)	36,928
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 64)	16,448
dense_1 (Dense)	(None, 2)	130

Total params: 72,898 (284.76 KB)
Trainable params: 72,898 (284.76 KB)
Non-trainable params: 0 (0.00 B)

Figure 5. Model summary of model-1

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 18, 32)	896
max_pooling2d (MaxPooling2D)	(None, 14, 9, 32)	0
conv2d_1 (Conv2D)	(None, 12, 7, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 3, 64)	0
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 64)	73,792
dense_1 (Dense)	(None, 2)	130

Total params: 93,314 (364.51 KB)
Trainable params: 93,314 (364.51 KB)
Non-trainable params: 0 (0.00 B)

Figure 6. Model summary of model-2

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 18, 64)	1,792
max_pooling2d (MaxPooling2D)	(None, 14, 9, 64)	0
conv2d_1 (Conv2D)	(None, 12, 7, 32)	18,464
max_pooling2d_1 (MaxPooling2D)	(None, 6, 3, 32)	0
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36,928
dense_1 (Dense)	(None, 2)	130

Total params: 57,314 (223.88 KB)
Trainable params: 57,314 (223.88 KB)
Non-trainable params: 0 (0.00 B)

Figure 7. Model summary of model-3

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 18, 64)	640
max_pooling2d (MaxPooling2D)	(None, 14, 9, 64)	0
conv2d_1 (Conv2D)	(None, 12, 7, 32)	18,464
max_pooling2d_1 (MaxPooling2D)	(None, 6, 3, 32)	0
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36,928
dense_1 (Dense)	(None, 64)	4,160
dense_2 (Dense)	(None, 2)	130

Total params: 60,322 (235.63 KB)
Trainable params: 60,322 (235.63 KB)
Non-trainable params: 0 (0.00 B)

Figure 8. Model summary of model-4

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 18, 64)	640
max_pooling2d (MaxPooling2D)	(None, 14, 9, 64)	0
conv2d_1 (Conv2D)	(None, 12, 7, 32)	18,464
max_pooling2d_1 (MaxPooling2D)	(None, 6, 3, 32)	0
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36,928
dense_1 (Dense)	(None, 100)	6,500
dense_2 (Dense)	(None, 2)	202

Total params: 62,734 (245.05 KB)
Trainable params: 62,734 (245.05 KB)
Non-trainable params: 0 (0.00 B)

Figure 9. Model summary of model-5

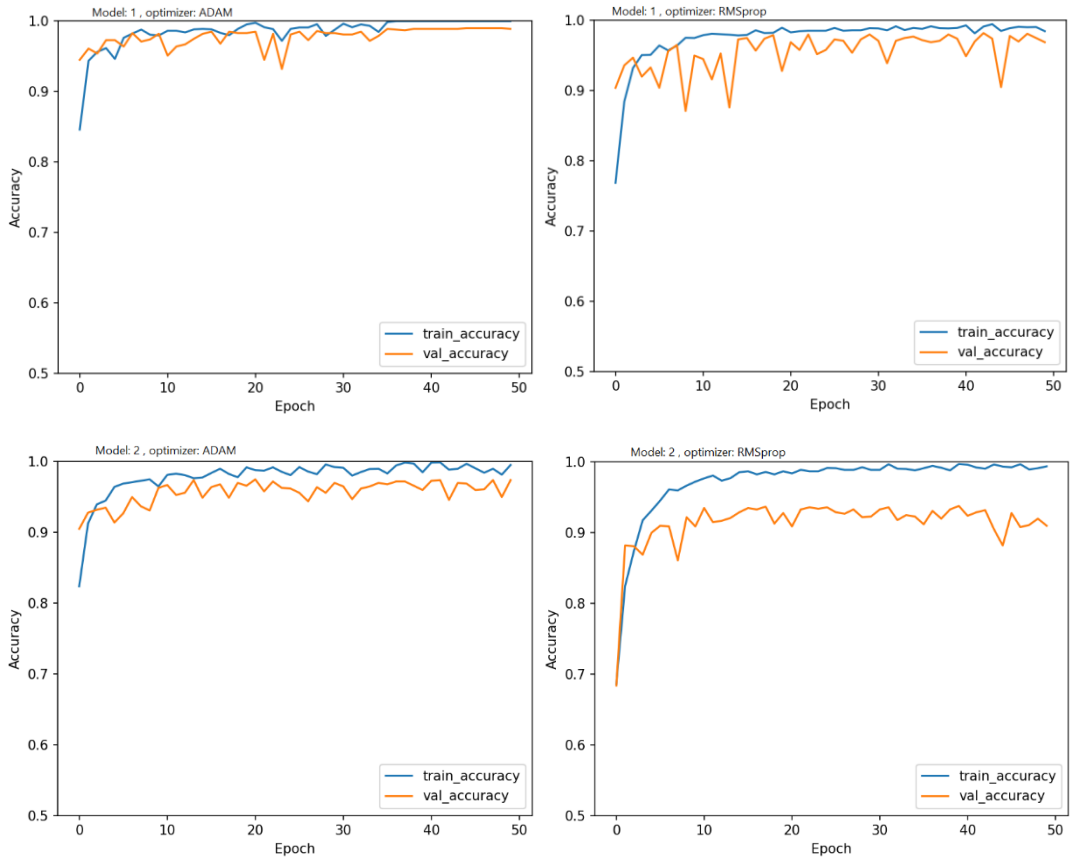


Figure 10. Some of training and validation accuracy

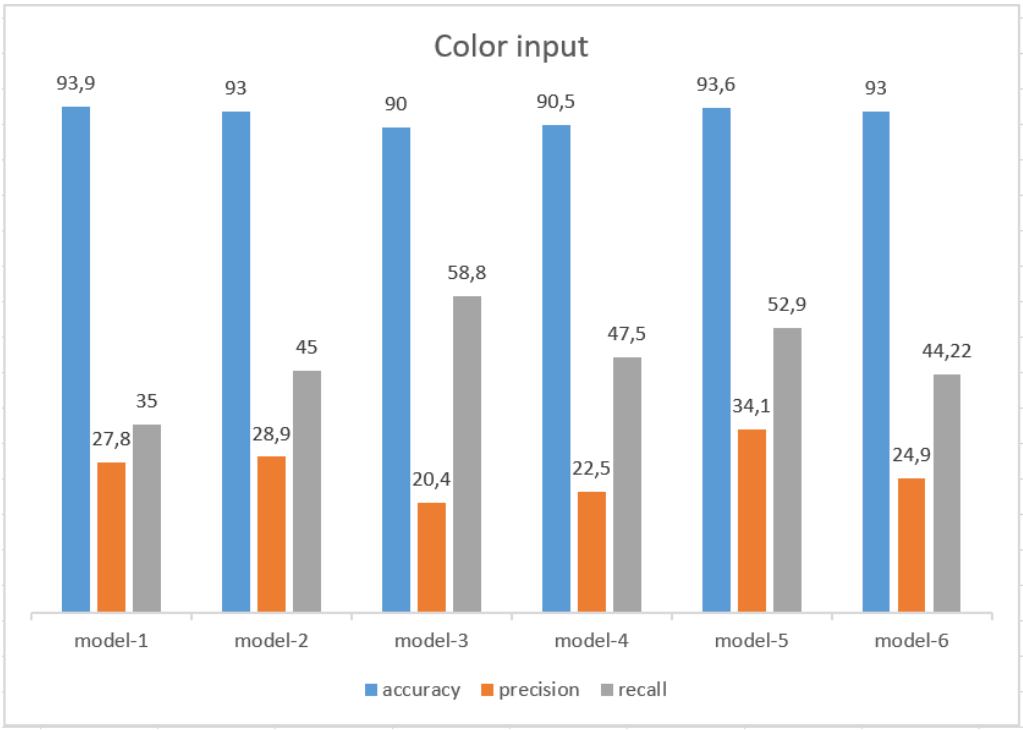


Figure 11. The performance of head detection, 30×20 pixels, using color input, **model-1**: model_1_adam, **model-2**: model_2_adam, **model-3**: model_3_adam, **model-4**: model_1_RMSprop, **model-5**: model_2_RMSprop, **model-6**: model_3_RMSprop,

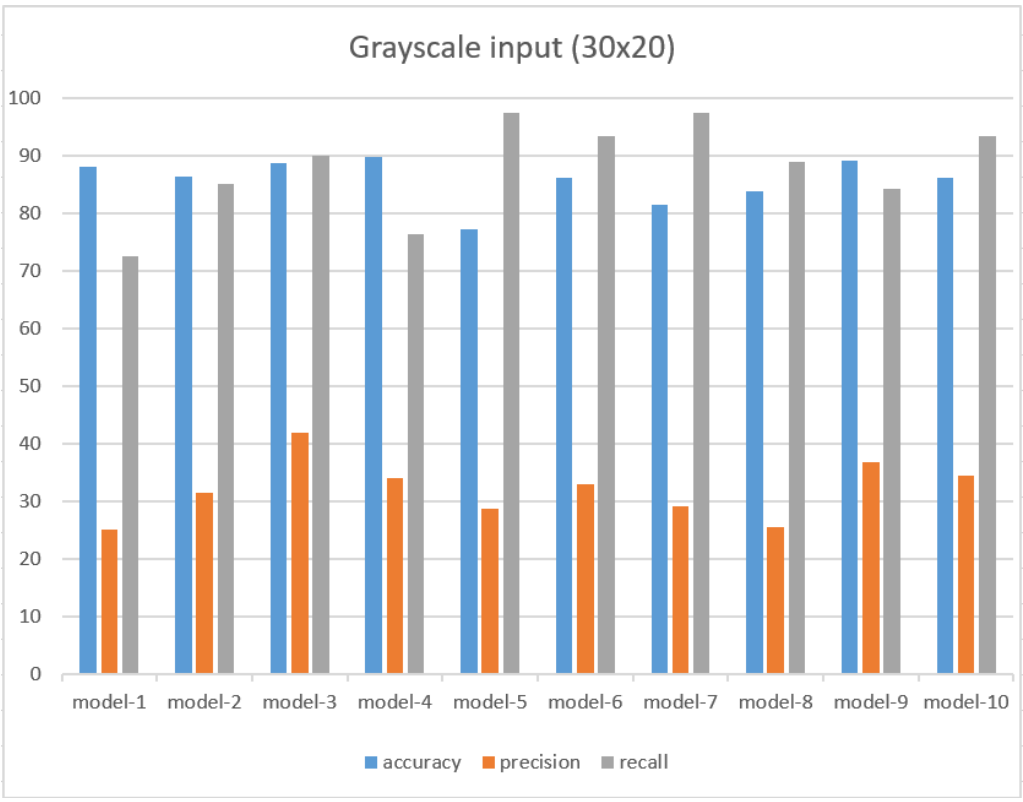


Figure 12. The performance of head detection, 30×20 pixels, using grayscale input, **model-1**: model_1_adam, **model-2**: model_2_adam, **model-3**: model_3_adam, **model-4**: model_1_RMSprop, **model-5**: model_2_RMSprop, **model-6**: model_3_RMSprop, **model**-7: model_4_adam, **model**-8: model_4_RMSprop, **model**-9: model_5_adam, **model**-10: model_5_RMSprop.

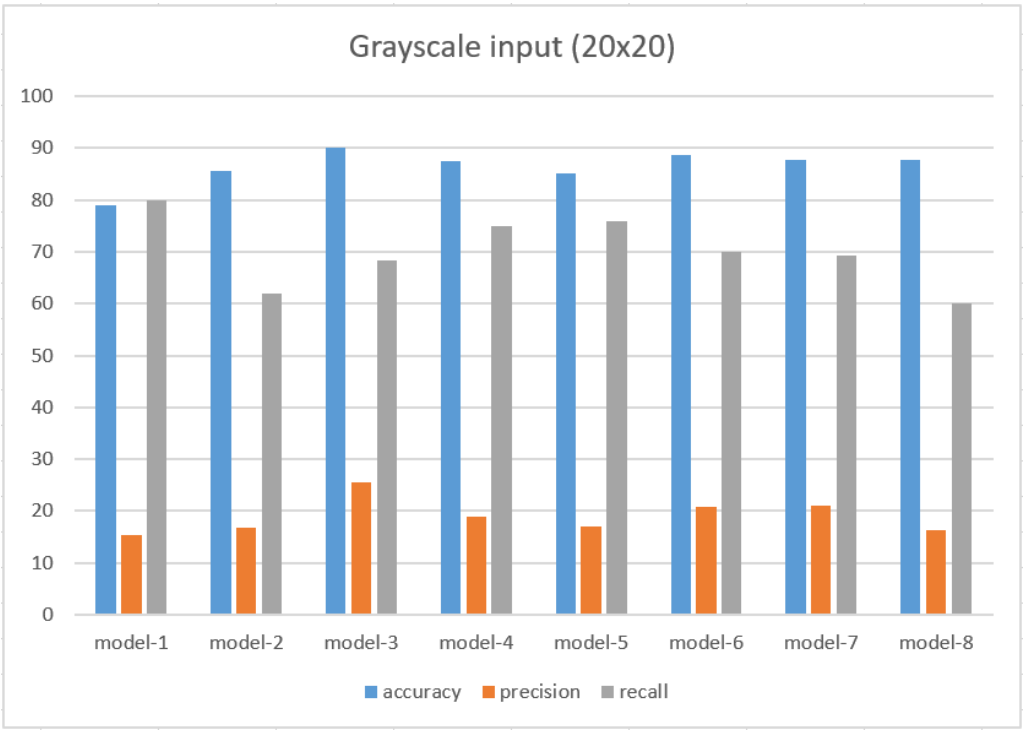


Figure 13. The performance of head detection, 20×20 pixels, using grayscale input, **model-1**: model_1_adam, **model-2**: model_2_adam, **model-3**: model_3_adam, **model-4**: model_1_RMSprop, **model-5**: model_2_RMSprop, **model-6**: model_3_RMSprop, **model**-7: model_4_adam, **model**-8: model_4_RMSprop.

To evaluate the detection performance, we perform the quantity as below:

Accuracy:

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

Precision:

$$precision = \frac{TP}{TP+FP} \quad (2)$$

Recall:

$$recall = \frac{TP}{TP+FN} \quad (3)$$

As we compare Figure 11 and Figure 12, grayscale images input are better than color images input. Precision value are low, due to the large number of false positive. From Figure 12 and Figure 13, training data with image size of 30×20 pixels are better than image size of 20×20 pixels.

IV. CONCLUSION

We have studied the performance of color and grayscale image input for CNN architecture for head detection. In training phase, all models have good validation accuracy, above 90%. However, in implementation for head detection, they have low precision value, due to the large number of false positive. The high performance of head detection is achieved by grayscale processing and image size of 30×20 pixels for training datasets.

Our future work is to apply pre-processing or other strategies to improve the precision value.

REFERENCES

[1]. Mudjirahardjo, P., Rahmansyah, A.G., Dianti, A.S.. The Performance of Convolutional Neural Network Architecture in Classification. International Journal of Computer Applications Technology and Research (IJCATR). Vol. 13, Issue 08. 2024. 115-122. ISSN: 2319-8656. DOI:10.7753/IJCATR1308.1011.

[2]. Mudjirahardjo, P., Rahmansyah, A.G., Dianti, A.S.. Head Classification based on Convolutional Neural Network. International Journal of Advanced Multidisciplinary Research and Studies (IJAMRS). Vol. 4, Issue 4. 2024. 982-989. ISSN: 2583-049x.

[3]. Alzubaidi, L., Zhang, J., Humaidi, A.J., et.al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. Journal of Big Data. 2021.

[4]. Yamashita, R., Nishio, M., Do, R.K.G. et al. Convolutional neural networks: an overview and application in radiology. Insights Imaging **9**, 2018, 611–629. https://doi.org/10.1007/s13244-018-0639-9

[5]. Hassan, E., Shams, M.Y., Hikal, N.A. et al. The effect of choosing optimizer algorithms to improve computer vision tasks: a comparative study. Multimed Tools Appl **82**, 2023, 16591–16633. https://doi.org/10.1007/s11042-022-13820-0

[6]. Abdulkadirov, R.; Lyakhov, P.; Nagornov, N. Survey of Optimization Algorithms in Modern Neural Networks. *Mathematics*, *11*, 2023, 2466. https://doi.org/10.3390/math11112466.

[7]. N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 2005, pp. 886-893 vol. 1, doi: 10.1109/CVPR.

[8]. Will Cukierski. CIFAR-10 - Object Recognition in Images. Kaggle. 2013. https://kaggle.com/competitions/cifar-10

[9]. Kingma, Diederik & Ba, Jimmy. Adam: A Method for Stochastic Optimization. International Conference on Learning Representations. 2014.

[10]. Gower RM, Loizou N, Qian X, Sailanbayev A, Shulgin E, Richtárik P SGD: general analysis and improved rates. In international conference on machine learning (pp. 5200-5209). 2019. PMLR.