Int. j. adv. multidisc. res. stud. 2024; 4(5):708-714

Received: 21-08-2024
Accepted: 01-10-2024

International Journal of Advanced Multidisciplinary Research and Studies

ISSN: 2583-049X

A Comparative Study on Input Format for Convolutional Neural Network based Head Detection

[1] Panca Mudjirahardjo, [2] Aqil Gama Rahmansyah, [3] Alya Shafa Dianti

[1, 2] Department of Electrical Engineering, Faculty of Engineering, Universitas Brawijaya, Malang, Indonesia

[3] Department of Statistics, Faculty of Mathematics and Natural Science, Universitas Brawijaya, Malang, Indonesia

Corresponding Author: **Panca Mudjirahardjo**

Abstract

In this research, we study and evaluate the performance of CNN based head detection with various input image. We evaluate grayscale, CLAHE and saliency map format as inputs to our CNN model. We use INRIA dataset for training and testing data. For training data, we use image size of 30×20 pixels. The experimental result shows saliency map is a good input and adam is a good optimizer for our CNN model. The experiment is conducted using programming language python and openCV library.

Keywords: CLAHE, Saliency Map, CNN, Head Detection, Optimizer

1. Introduction

Head detection is one of task in computer vision. The objective of head detection is for a smart monitoring system, both indoors and outdoors. Head detection and orientation estimation are a vital component in the intention recognition of pedestrians. This research still has challenges, due to the complexity of human poses, background, lighting conditions, occlusions and camera view-points. Head detection may be more demanding than face recognition and pedestrian detection in the scenarios where a face turns away or body parts are occluded in the view of a sensor, but locating people is needed.

Bin Li *et al* [1] captured the scene and detected human head from top view. They proposed a novel people counting method based on head detection and tracking to evaluate the number of people who move under an over-head camera. There were four main parts in the proposed method: foreground extraction, head detection, head tracking, and crossing-line judgment. The proposed method first utilized an effective foreground extraction method to obtain foreground regions of moving people, and some morphological operations were employed to optimize the foreground regions. Then it exploited a LBP feature based Adaboost classifier for head detection in the optimized foreground regions. After head detection was performed, the candidate head object was tracked by a local head tracking method based on Meanshift algorithm. Based on head tracking, the method finally used crossing-line judgment to determine whether the candidate head object will be counted or not. Experiments show that their method can obtain promising people counting accuracy about 96% and acceptable computation speed under different circumstances.

Eike Rehder, *et al.* [2] proposed a novel framework to detect highly occluded pedestrians and estimate their head orientation. Detection was performed for pedestrian's heads only. For this they employed a part-based classifier with HOG/SVM combinations. Head orientations were estimated using discrete orientation classifiers and LBP features. Results were improved by leveraging orientation estimation for head and torso as well as motion information. The orientation estimation was integrated over time using a Hidden Markov Model. From the discrete model they obtained a continuous head orientation. They evaluated their approach on image sequences with ground truth orientation measurements.

Tuan-Hung Vu, *et al.* [3] focused on detecting human heads in natural scenes. Starting from the recent local R-CNN object detector, they extended it with two types of contextual cues. First, they leveraged person-scene relations and proposed a Global CNN model trained to predict positions and scales of heads directly from the full image. Second, they explicitly modeled pairwise relations among objects and trained a Pairwise CNN model using a structured-output surrogate loss. The Local, Global and Pairwise models were combined into a joint CNN framework. To train and test their full model, they introduced a large dataset composed of 369, 846 human heads annotated in 224, 740 movie frames. They evaluated their method and

demonstrated improvements of person head detection against several recent baselines in three datasets.

Siyuan Chen, *et al.* [4] introduced an efficient head detection approach for single depth images at low computational expense. First, a novel head descriptor was developed and used to classify pixels as head or non-head. They used depth values to guide each window size, to eliminate false positives of head centers, and to cluster head pixels, which significantly reduced the computation costs of searching for appropriate parameters. High head detection performance were achieved in experiments – 90% accuracy for our dataset containing heads with different body postures, head poses, and distances to a Kinect2 sensor, and above 70% precision on a public dataset composed of a few daily activities, which is higher than using a head-shoulder detector with HOG feature for depth images.

Dexhi Peng, *et al.* [5] presented a method that can accurately detect heads especially small heads under the indoor scene. To achieve this, they proposed a novel method, Feature Refine Net (FRN), and a cascaded multi-scale architecture. FRN exploits the multi-scale hierarchical features created by deep convolutional neural networks. The proposed channel weighting method enables FRN to make use of features alternatively and effectively. To improve the performance of small head detection, they proposed a cascaded multi-scale architecture which has two detectors. One called global detector was responsible for detecting large objects and acquiring the global distribution information. The other called local detector was designed for small objects detection and made use of the information provided by global detector. Due to the lack of head detection datasets, they had collected and labeled a new large dataset named SCUT-HEAD which includes 4405 images with 111251 heads annotated. Experiments show that their method had achieved state-of-the-art performance on SCUT-HEAD.

Muhammad Saqib, *et al.* [6] detected human heads in natural scenes acquired from a publicly available dataset of Hollywood movies. In this work, we had used state-of-the-art object detectors based on deep convolutional neural networks. These object detectors include region-based convolutional neural networks using region proposals for detections. Also, object detectors that detect objects in the single-shot by looking at the image only once for detections. They had used transfer learning for fine-tuning the network already trained on a massive amount of data. During the fine-tuning process, the models having high mean Average Precision (mAP) were used for evaluation of the test dataset.

Yijing Wang, *et al.* [7] developed a simple effective proposal-based human head and body detection framework in crowded scenes. Human heads were too small for detectors to locate and human bodies were frequently occluded in the crowds, which required more robust location capability of detectors. To tackle the issues above, they proposed a head-body correlation module to utilize the location prior knowledge of human body and human head. Compared with Faster R-CNN, their approach can improve the Average Precision (AP) gains for human body and head detection by 2.15% and 2.52% on the challenging CrowdHuman dataset.

Xiyang Dai, *et al.* [8] presented a novel dynamic head framework to unify object detection heads with attentions. By coherently combining multiple self-attention mechanisms between feature levels for scale awareness, among spatial locations for spatial-awareness, and within output channels for task-awareness, the proposed approach significantly improved the representation ability of object detection heads without any computational overhead. Further experiments demonstrated that the effectiveness and efficiency of the proposed dynamic head on the COCO benchmark. With a standard ResNeXt-101- DCN backbone, they largely improved the performance over popular object detectors and achieved a new state-of-the-art at 54.0 AP. Furthermore, with latest transformer backbone and extra data, they can push current best COCO result to a new record at 60.6 AP.

2. The Proposed Study

In this section, we briefly explain the proposed study to evaluate the performance of various input format for convolutional neural network. We evaluate grayscale format, Contrast Limited Adaptive Histogram Equalization (CLAHE) format and saliency map as an input image. The study method is shown in Fig 1. Our architecture in this study use the architecture in Fig 2. The architecture has been evaluated in [11-12].

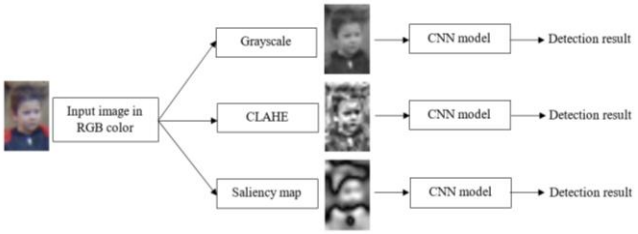


Fig 1: The proposed method of this study

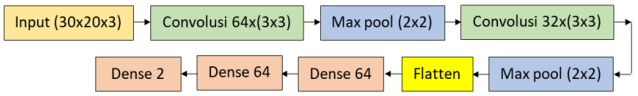


Fig 2: The CNN model we used [12]

3. Convolutional Neural Network

A Convolutional Neural Network (CNN) is a type of artificial neural network designed primarily for processing structured grid data, such as images. Here's a brief overview of its key components and how it works:

Key Components

1. **Convolutional Layers:** These layers apply convolutional filters (kernels) to the input data. Each filter detects specific features such as edges or textures. As the filter slides over the input image, it produces feature maps that represent the presence of these features.
2. **Activation Functions:** After convolution, activation functions like ReLU (Rectified Linear Unit) introduce non-linearity to the model, helping it learn more complex patterns.
3. **Pooling Layers:** These layers reduce the spatial dimensions (width and height) of the feature maps while retaining the most important information. Common pooling operations include max pooling (taking the maximum value in a region) and average pooling.
4. **Fully Connected Layers:** After several convolutional and pooling layers, the network typically includes one or more fully connected layers that perform classification or regression based on the extracted features.

5. **Dropout Layers:** To prevent overfitting, dropout layers randomly "drop" (set to zero) a fraction of the neurons during training, which helps the network generalize better to new, unseen data.

How It Works

1. **Feature Extraction:** CNNs automatically learn and extract features from the input data. For an image, this means learning to detect edges, textures, and more complex structures as you go deeper into the network.
2. **Hierarchical Learning:** Lower layers in the network might learn simple features like edges, while higher layers combine these features to detect more complex structures, such as shapes or objects.
3. **Classification/Regression:** After extracting features, CNNs use fully connected layers to classify the image into categories or predict values if used for regression tasks.

Applications

CNNs are widely used in various fields:

- **Image Recognition:** Identifying objects, people, or scenes in images.
- **Object Detection:** Locating objects within an image and classifying them.
- **Semantic Segmentation:** Assigning a class to each pixel in an image.
- **Video Analysis:** Recognizing actions or events in video frames.
- **Medical Imaging:** Analyzing medical scans for disease detection or diagnosis.

4. Training phase

The training phase in machine learning is a crucial part of developing a model that can make accurate predictions or decisions based on data. Here's a detailed look at what happens during the training phase:

Steps in the Training Phase

1. Data Preparation
   - **Data Collection:** Gather the dataset that will be used for training. This could be from various sources like databases, web scraping, or existing datasets.
   - **Data Cleaning:** Handle missing values, remove duplicates, and correct errors to ensure the data is of high quality.
   - **Data Splitting:** Divide the dataset into training, validation, and test sets. Typically, the training set is used to train the model, the validation set is used to tune hyperparameters, and the test set is used to evaluate the model's performance.

2. Model Initialization
   - **Choosing a Model:** Select an appropriate model or algorithm based on the problem type (e.g., linear regression, decision tree, neural network).
   - **Initializing Parameters:** Set initial values for the model's parameters. For complex models like neural networks, these are often initialized randomly.

3. Forward Pass
   - **Input Data:** Feed a batch of training data into the model.
   - **Prediction:** The model processes the input data through its layers (in the case of neural networks) and generates predictions or outputs.

4. Loss Calculation
   - **Loss Function:** Compute the loss (or error) by comparing the model's predictions with the actual target values using a loss function (e.g., mean squared error, cross-entropy loss).
   - **Objective:** The goal is to minimize this loss function.

5. Backward Pass (Backpropagation in Neural Networks)
   - **Gradient Calculation:** Calculate the gradients of the loss function with respect to each model parameter using techniques like gradient descent.
   - **Parameter Update:** Adjust the model parameters based on the gradients to reduce the loss. This involves using an optimizer (e.g., SGD, Adam) to apply updates.

6. Iteration
   - **Epochs:** Repeat the forward pass, loss calculation, and backward pass for multiple epochs (iterations over the entire training dataset).
   - **Mini-batch Processing:** For large datasets, data is often processed in smaller mini-batches rather than all at once.

7. Validation
   - **Hyperparameter Tuning:** Use the validation set to tune hyperparameters (e.g., learning rate, number of layers) and make adjustments to improve performance.
   - **Model Evaluation:** Periodically evaluate the model on the validation set to monitor its performance and ensure it is not overfitting.

8. Regularization
   - **Techniques:** Apply regularization techniques (e.g., dropout, L2 regularization) to prevent overfitting and improve generalization.
   - **Early Stopping:** Monitor validation performance and stop training if performance on the validation set starts to degrade.

9. Model Saving
   - **Checkpointing:** Save the model parameters and state at different points during training, especially after significant improvements.
   - **Best Model:** Save the best performing model based on validation metrics.

Key Concepts

- **Overfitting:** The model performs well on training data but poorly on validation/test data. This often means the model has learned noise in the training data.
- **Underfitting:** The model performs poorly on both training and validation data, indicating it is too simple to capture the underlying patterns in the data.
- **Learning Rate:** Determines the size of the steps taken during parameter updates. Too high a learning rate can cause the model to overshoot minima, while too low a rate can lead to slow convergence.

- **Epochs:** The number of times the entire training dataset is passed through the model. More epochs can lead to better training, but also increase the risk of overfitting.
- **Batch Size:** The number of training examples used in one iteration of model updates. A larger batch size can stabilize training but requires more memory.

The training phase is where a machine learning model learns from data by adjusting its parameters to minimize a loss function. It involves data preparation, model initialization, forward and backward passes, and iteration with validation. Regularization techniques are used to enhance the model's ability to generalize to new, unseen data. Proper management of this phase is essential for developing a robust and effective machine learning model.

In this study, we have 2 classes and put the training data in directory:

```
D:\Riset\[1] DATA_image\INRIA - ku\datasets_2000x2\
        Head_OK\
                head (1).png
                head (2).png
                ---
                ---
        Head_NG
                neg (1).png
                neg (2).png
                ---
                ---
```

Program-1:

```
print('CNN training 30x20 Grayscale, part 3 ..\n'*5)
print("ARZETI_Doyoubi,24.08.2024; 08:20")
print("Panca" +
     " Mudjirahardjo")
print("")
print("====================================")

print('')
optim = input('Optimizer: (1)ADAM, (2)RMSprop : ')
print('')
ep = input('The number of 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():
    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

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

data_dir = "D:\Riset\[1] DATA_image\INRIA -
ku\datasets_2000x2"

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)

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

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

model.compile(
  optimizer=optim,
   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_l
ogits=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')
```

5. Head Detection

Program-2:

```
print('CNN head detection, part 1 ..\n'*5)
print("ARZETI_Nichiyoubi,18.08.2024; 12:43")
print("Panca" +
      " Mudjirahardjo")
print("")
print("==================================")

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

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

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

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

oriIMG = cv.imread("D:\Program\output image\person_1.jpg")

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

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

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

model = models.load_model('D:\Program\python 3.11.5\Training model\myModel.keras')

print('')

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

for r in range(0,h,15):
    for c in range(0,w,10):
        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]

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

cv.imshow('image',oriIMG)
cv.waitKey(0)
```

6. The Experimental Result

In this section, the experimental procedure and result are briefly explain. This experiment is performed using programming language python and openCV library. Code program of training phase and head detection are written in Program-1 and Program-2, respectively. To evaluate the performance input format, we use scenes as shown in Fig 3. To create CLAHE image and saliency map are written in Program-3 and Program-4, respectively.

Fig 3: Some of the scenes used for this experiment [9]

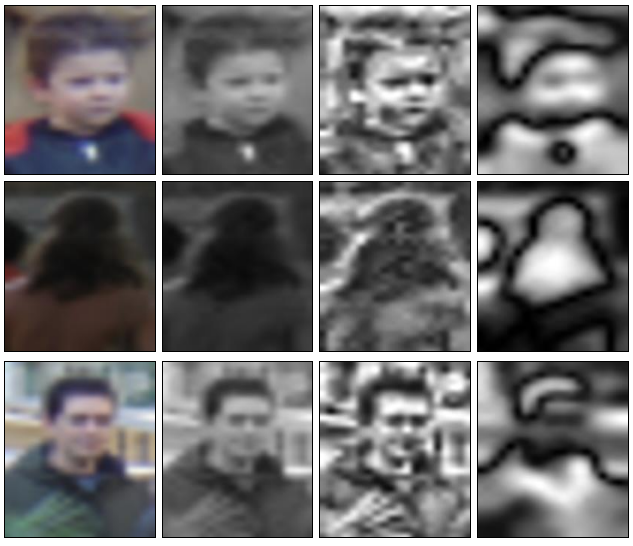
Program-3:

```
# -- Applying CLAHE process --
clahe = cv.createCLAHE(clipLimit=5)
imgCLAHE = clahe.apply(img)
```

Program-4:

```
# -- SALIENCY map --
sal= cv.saliency.StaticSaliencyFineGrained_create()
(success, saliencyMap) = sal.computeSaliency(img)
saliencyMap = (saliencyMap * 255).astype("uint8")
```

Some of training data in various format are depicted in Fig 4.



(a) (b) (c) (d)

Fig 4: Some of input format (a) original image [9] (b) grayscale image (c) CLAHE image (d) saliency image

To evaluate the performance of input image format into CNN, we use the quantities are below:
Accuracy:

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

Precision:

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

Recall:

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

Where:

***TP*:** True positive, i.e. head is detected as head,
***TN*:** True negative, i.e. non-head is detected as non-head,
***FP*:** False positive, i.e. non-head is detected as head,
***FN*:** False negative, i.e. head is detected as non-head.

The detection result is shown in Table 1 and Fig 5. As Table 1 and Fig 5 show good head detection performance using saliency map as input image. The precision value is low, because there are many false positives.

Table 1: Performance of head detection

Input format	Optimizer	Accuracy	Precision	Recall
Grayscale	Adam	0.814	0.292	0.975
	RMSprop	0.838	0.255	0.888
CLAHE	Adam	0.609	0.159	0.900
	RMSprop	0.698	0.193	0.933
Saliency	**Adam**	**0.870**	**0.350**	**0.983**
	RMSprop	0.781	0.279	1.00

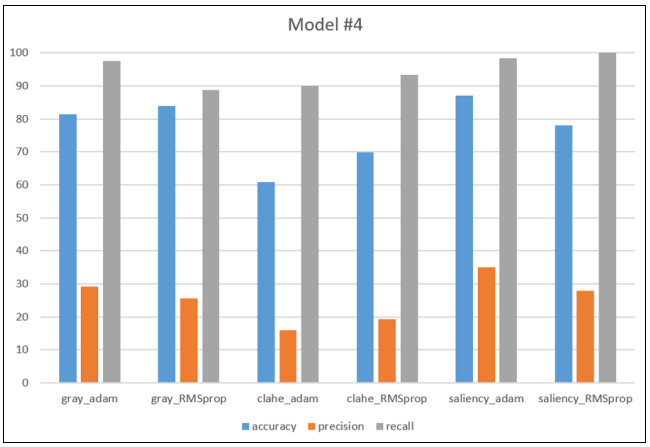


Fig 5: The performance of different input format and optimizer

7. Conclusion

From the above study, we evaluate the performance of CNN based head detection with various input image. We evaluate grayscale, CLAHE and saliency format as inputs to our CNN model. As shown in Table 1 and Fig 5, we conclude that saliency map is a good input and adam is a good optimizer for our CNN model.
Our future work is to observe other methods to achieve the best performance, namely increasing the precision value.

8. References

1. Li B, Zhang J, Zhang Z, Xu Y. A People Counting Method Based on Head Detection and Tracking. 2014 International Conference on Smart Computing, Hong Kong, China, 2014, 136-141. Doi: 10.1109/SMARTCOMP.2014.7043851.
2. Rehder E, Kloeden H, Stiller C. Head detection and orientation estimation for pedestrian safety, 17th International IEEE Conference on Intelligent Transportation Systems (ITSC), Qingdao, China, 2014, 2292-2297. Doi: 10.1109/ITSC.2014.6958057.
3. Vu T-H, Osokin A, Laptev I. Context-Aware CNNs for Person Head Detection, 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 2015, 2893-2901. Doi: 10.1109/ICCV.2015.331.
4. Siyuan Chen F, Bremond Hung Nguyen, Thomas H. Exploring depth information for head detection with depth images, 2016 13th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), Colorado Springs, CO, 2016, 228-234. Doi: 10.1109/AVSS.2016.7738060
5. Dezhi Peng, Zikai Sun, Zirong Chen, Zirui Cai, Lele Xie, Lianwen Jin. Detecting Heads using Feature Refine Net and Cascaded Multi-scale Architecture. 2018 24th International Conference on Pattern Recognition (ICPR), 2018, 2528-2533.
6. Saqib M, Khan SD, Sharma N, Blumenstein M. Person Head Detection in Multiple Scales Using Deep Convolutional Neural Networks, 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 2018, 1-7. Doi: 10.1109/IJCNN.2018.8489367.
7. Wang Y, Zhang L, Zuo Z, Cheng X. Head-Body Correlation for Robust Crowd Human Detection, 2021 40th Chinese Control Conference (CCC), Shanghai, China, 2021, 7282-7287. Doi: 10.23919/CCC52363.2021.9550747.
8. Xiyang Dai, Yinpeng Chen, Bin Xiao, Dongdong Chen, Mengchen Liu, Lu Yuan, Lei Zhang. Dynamic Head: Unifying Object Detection Heads with Attentions. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, 7369-7378.
9. Dalal N, Triggs B. 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; 1:886-893. Doi: 10.1109/CVPR.
10. Mudjirahardjo P, Rahmansyah AG, Dianti AS. The Performance of Convolutional Neural Network Architecture in Classification. International Journal of Computer Applications Technology and Research (IJCATR). 2024; 13(08):115-122. ISSN: 2319-8656. Doi: 10.7753/IJCATR1308.1011.
11. Mudjirahardjo P, Rahmansyah AG, Dianti AS. Head Classification based on Convolutional Neural Network. International Journal of Advanced Multidisciplinary Research and Studies (IJAMRS). 2024; 4(4):982-989. ISSN: 2583-049x.
12. Mudjirahardjo P, Rahmansyah AG, Dianti AS. The performance of color and grayscale image input in convolutional neural network based head detection. International Refereed Journal of Engineering and

Science (IRJES). 2024; 13(5):45-62. E-ISSN: 2319-183x.
13. Venkatesh S, John De Britto C, Subhashini P, Somasundaram K. Image Enhancement and Implementation of CLAHE Algorithm and Bilinear Interpolation. Cybernetics and systems: An International Journal, 2022.