HO CHI MINH CITY

UNIVERSITY OF TECHNOLOGY AND EDUCATION

FACULTY OF HIGH QUALITY TRAINING

REPORT OF SUBJECT

MACHINE VISION

Lecturer: PhD. BUI HA DUC	
Name:	**ID:**
Nguyen Huu Dang Khoa	**20146218**
Nguyen Cong Danh	**20146231**
Class: SEAC225929E_22_2_03CLC	

Ho Chi Minh, May 2023

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Machine Vision

Machine vision is a technology field in the field of Artificial Intelligence intending to handle tasks requiring visual supervision in industrial production, entertainment - sports activities, etc. Examine and analyze image data.

This field has opened up many new aspects and facilitated the development of Artificial Intelligence. Machine vision will greatly assist humans in tracking, monitoring, etc. activities to be more accurate and effective.

Approach Machine Vision is not an easy process, requiring a long approach to grasp the essence of this technology..

1.2 Deep Learning

Machine vision is simply the processing of images in simple tasks, to meet more difficult requirements such as: “determining the trajectory of a flying object”, “counting the number of vehicles circulating on a plane”, … then we need to give it an artificial brain.

This brain is formed by building the data model for the request. We will perform each data search, train the model, and then combine the model with Machine Vision to produce a task that meets the problem to be solved.

CHAPTER 2: COLLECT DATA FEATURES

2.1 Collection

Data preparation is an important part. The training data plays a decisive role in determining whether the model works well or not. The world's major intelligent AI models are trained from abundant data sources. The data helps to extract the features of objects, so when the model approaches new data, it will accurately judge the object. What is the requirement of that problem?

In this object recognition problem, the collected data consists of three compressed data files. The first file is the file "train.zip" which is the raw data with different sizes used to train the model. in which there are two folders, the first folder contains the data of the basketball ball named "1" and the other folder is the image

without the ball, it can be human hands, feet, tables and chairs and other objects. The orange object matches the ball and is placed in the “0” folder for removal because it is not typical of a basketball. The recognized basketball will have an orange color, a black border, and a label. In the train data, my colleagues and I also based on those features that enrich the data by rolling the basketball, bouncing, the result is that the ball will be blurred, blurred and sometimes near and far from the camera. At this time, the data in the training file has covered the entire experimental area including all weather and lighting conditions.

The second file is "validation.zip", which will also be classified and labeled like the "train.zip" file, but its data does not belong to the training data. This means that there will still be two folders "0" and "1", but these data are new. The validation set helps to verify the accuracy of the training model.

Finally, there is the "test.zip" file, which includes images of basketballs and images to be mixed. This file will be used after the model has been trained. To determine if the trained model is good, we use new data to approach the training model. In this dataset, data must be standardized like the model's data, as we will discuss later. It is also classified into 2 folders for necessary cases.

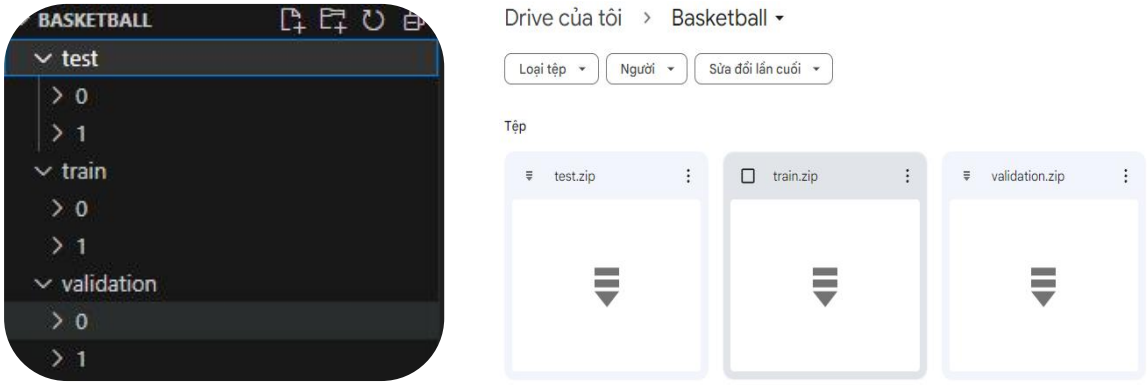


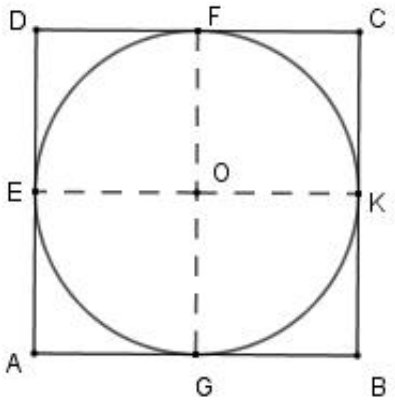
Figure 2.1: Data files

2.2 Analyze collected data

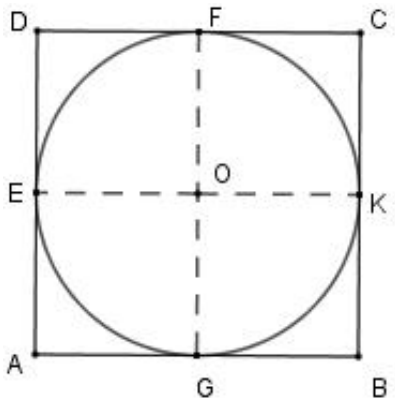
Training data is important, so let's analyze it a bit because when training and using it, the data in the test file and the data in the training file must be somewhat similar.

In the training file, we see that it will have the form of a circle inscribed in a square, with 4 corners being the color of the surrounding environment, and in the middle of the square is an orange basketball that is rotated in many different directions. This has a significant impact on image processing, which means that after detecting it by traditional methods such as finding thresholds in binary images or detecting it by a HSV color frame, we must crop it into an image similar to the one in the training section so that the classification can be more accurate.

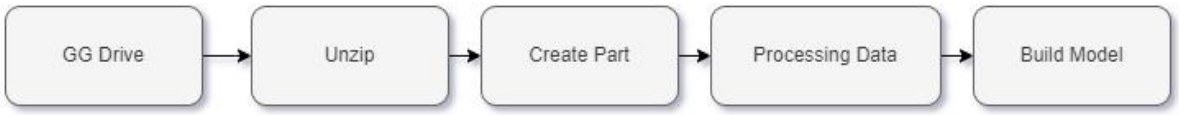
In case of correct identification:



In case of false identification:



CHAPTER 3: TRAINING MODEL



3.1. Building data Pipeline from GG Colab

The reason for building a data pipeline is that we can directly manage data on Google Drive and download data directly to train the model. Another reason is that we can link it through Google Colab to use it to train the model with GPU runtime. Google's GPU will help reduce the training time significantly and save computer processing resources.

To do this, we need to upload data to Google Colab and create paths for data folders.

Reference code:

Google Drive Data

```
[54] from google.colab import drive
     drive.mount('/content/drive')

     Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

[55] !ls /content/drive/MyDrive/Basketball

     test.zip  train.zip  validation.zip

! unzip /content/drive/MyDrive/Basketball/train.zip
! unzip /content/drive/MyDrive/Basketball/test.zip
! unzip /content/drive/MyDrive/Basketball/validation.zip
```

Figure 3.1: Decompress the data

Test data:

```
for i in range(2):
  !ls test/{i}| wc -l
for i in range(2):
  !ls validation/{i}| wc -l
!ls test
for i in range(2):
  !ls train/{i}| wc -l
```

Figure 3.2: Test data

Create path:

```
# Part data train
train_root = Path('train/')
filelist_Training = tf.data.Dataset.list_files(str(train_root/'*/*'))

# Part data test
test_root = Path('test/')
filelist_Testing = tf.data.Dataset.list_files(str(test_root/'*/*'))

# Part data validation
val_root = Path('validation/')
filelist_Validation = tf.data.Dataset.list_files(str(val_root/'*/*'))
```

Figure 3.3: Create a path to the data

3.2. Processing Data

Proceed to process the input image and simultaneously label each class. In the preprocessing function, we will read it as a color image (channels = 3), then convert it to grayscale. Next, normalize the image to a value between 0 and 1, then resize the image to the size of (28,28). Finally, reshape the image into a vector.

```
[70] #Preprocessing
     def preprocessing(file_path):
       #Readfile
       img1 = tf.io.read_file(file_path)
       img1 = tf.image.decode_jpeg(img1,channels=3) #Read Color image
       img1 = tf.image.rgb_to_grayscale(img1, name=None) # convert to gray scale
       #Transform
       img1 = tf.image.convert_image_dtype(img1,tf.float32) # convert uint8 to float and normalize [0,1]
       #Resize image
       img1 = tf.image.resize(img1,[img_width,img_height])
       #-4. Reshape -> vector hàng
       img1 = tf.reshape(img1,(28,28,1))
       #Get image lable
       label = get_label(file_path) #sự dụng hàm test label vừa tạo
       #Return
       return img1,label#Preprocessing
```

```
[71] # Make Label
     def get_label(file_path):
       parts = tf.strings.split(file_path, '/')
       if parts[-2] == "0":
         labels=[1,0]
       else:
         labels=[0,1]
       return tf.convert_to_tensor(labels)
```

Figure 3.4: Data Normalization and Labeling

Building the training data:

```
[73] #build data training
     train_dataset = filelist_Training.shuffle(datatrain_size)
     train_dataset = train_dataset.map(preprocessing,num_parallel_calls=tf.data.AUTOTUNE) # Process each eleme
     train_dataset = train_dataset.batch(batch_size)
     train_dataset = train_dataset.prefetch(2) # chia luõn data
```

Figure 3.5: The training data

3.3. Building Model

```
# Build CNN model using function API
# Convolutional Neural Network

model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(img_height,img_width,1)))
model.add(layers.Conv2D(10,5, strides=(1,1), padding='valid', activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2), strides=2))
model.add(layers.Conv2D(20,5,strides=(1,1), padding='valid', activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2), strides=2))
model.add(layers.Flatten())
model.add(layers.Dropout(0.25))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dense(2))
model.add(layers.Softmax())

model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy']) # Compile model
model.summary()

# Draw Graph
tf.keras.utils.plot_model(model,'model.png',show_shapes=True)

early_stopping = EarlyStopping(monitor='val_loss', patience=3)
```

Figure 3.6: Building models through data filtering layers

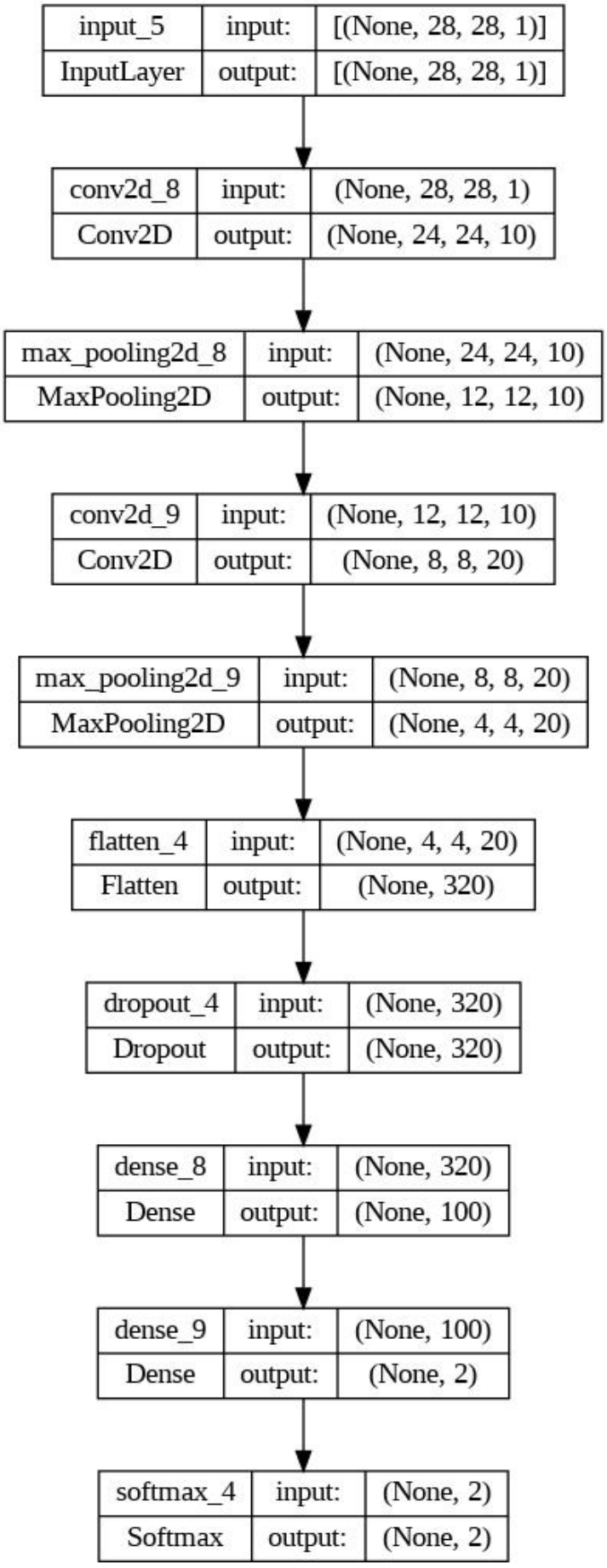


Figure 3.7: Model tree

3.4. Training and Evaluate Model

```
# Train model
history = model.fit(train_dataset, epochs=30, validation_data=test_dataset_validation, callbacks=[early_stopping])

# Get training and test loss histories
training_loss = history.history['loss']
test_loss = history.history['val_loss']

# Create count of the number of epochs
epoch_count = range(1, len(training_loss) + 1)

# Visualize loss history
plt.plot(epoch_count, training_loss, 'r--')
plt.plot(epoch_count, test_loss, 'b-')
plt.legend(['Training Loss', 'Test Loss'])
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()

# Get training and test accuracy histories
training_accuracy = history.history['accuracy']
test_accuracy = history.history['val_accuracy']

# Visualize accuracy history
plt.plot(epoch_count, training_accuracy, 'r--')
plt.plot(epoch_count, test_accuracy, 'b-')
plt.legend(['Training Accuracy', 'Test Accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy Score')
plt.show()

model.save('CNN_Bask_Rec.h5')
```

Figure 3.8: Training and Evaluate

CHAPTER 4: APPLYING MODEL FOR MACHINE VISION

4.1 Object detection

Looking at the requirements of the task which is to Detect and Track the basketball, the first thing we need to do is to write a program to detect the basketball based on the trained model. If the detected object is a basketball, we will proceed to track that basketball. During the tracking process, if the ball goes out of the frame or the tracking algorithm loses it, we will perform a ball detection algorithm and continue the tracking.

```
frameblur = cv.GaussianBlur(frame, (5, 5), 1)
imgColor, mask =mycolor.update(frameblur, hsvVal)
contours, hierarchy = cv.findContours(mask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
for cnt in contours:
    if cv.contourArea(cnt) > 1500:
        #crop roi
        x,y,w,h=cv.boundingRect(cnt)
        roi=imgColor[y:y+h, x:x+w]
        #resize to 28x28
        roi=cv.resize(roi,(28,28))
        #roi = load_img(roi,target_size=(28,28))
        cv.imshow('Basket',roi)
        # normalize range(0,1)
        roi = img_to_array(roi)
        roi=roi.reshape(3,28,28,1)
        roi = roi.astype('float32')
        roi =roi/255
        #predict
        while state == 0:
            result=np.argmax(model.predict(roi),axis=1)
```

Figure 4.1: Ball detection algorithm

First of all, to talk about the basketball detection algorithm, we must ensure that we have obtained the trained basketball model that we need to manipulate. The processing steps are as follows, in which we use basketball as an example:

Step 1: Using the CVZone library to determine the HSV color range of the ball in the input video clip environment.

```
mycolor = ColorFinder(False)
hsvVal = {'hmin': 4, 'smin': 100, 'vmin': 60, 'hmax': 17, 'smax': 255, 'vmax': 255}
```

Figure 4.2: Determining the HSV of the ball

Step 2: Filter noise in each frame of the clip and use the function imgColor, mask = mycolor.update(frameblur, hsvVal) to obtain each image of the ball by sending the HSV values to the function (imgColor is the ball image in RGB mode, mask is the binary image of the ball).

Step 3: Use the FindContours function to identify the area where the ball appears and then crop that part of the image. Process the image through steps such as resizing to 28x28 and normalizing the image to fit the input of the trained model.

```
contours, hierarchy = cv.findContours(mask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
for cnt in contours:
    if cv.contourArea(cnt) > 1500:
        #crop roi
        x,y,w,h=cv.boundingRect(cnt)
        roi=imgColor[y:y+h, x:x+w]
        #resize to 28x28
        roi=cv.resize(roi,(28,28))
        #roi = load_img(roi,target_size=(28,28))
        cv.imshow('Basket',roi)
        # normalize range(0,1)
        roi = img_to_array(roi)
        roi=roi.reshape(3,28,28,1)
        roi = roi.astype('float32')
        roi =roi/255
```

Figure 4.3: Processing crop images for inclusion in the model

Step 4: Predict whether the cropped orange part is a basketball or not. If it is not, continue predicting in subsequent frames. If it is, start tracking the ball.

```
#predict
while state == 0:
    result=np.argmax(model.predict(roi),axis=1)
    if result[0] == 1:
        state = 0
    elif result[0] == 0:
        ret = tracker.init(frame, (x, y, w, h))
        state=1
        if state == 1:
            break
```

Figure 4.4: Object prediction

4.2. Object tracking

After identifying the object as a ball, we will perform tracking as follows:

Step 1: After identifying the ball, the "state" variable will change its value from 0 to 1. The code line "ret = tracker.init(frame, (x, y, w, h))" is used to pass the coordinates of the tracked region to the tracker.init function.

```
elif result[0] == 0:
    ret = tracker.init(frame, (x, y, w, h))
    state=1
```

Figure 4.5: Determine the coordinates of the area to be Tracking

Step 2: Create a point to draw the Tracking BoundingBox. The function "tracker.update()" is used to update the coordinate points that change in each frame,

and if the object is identified, the BoundingBox will be drawn. If the object is lost, the "state" variable will return to 0 and the ball tracking process will restart.

```
ret, obj = tracker.update(frame)
if ret:
        p1 = (int(obj[0]),int(obj[1]))
        p2 = (int(obj[0] + obj[2]),int(obj[1] + obj[3]))
        cv.rectangle(frame, p1,p2, (255,0,0), 2, 1)
else:
    state = 0
```

Figure 4.6: Update coordinates and draw Bounding Box

Through section 4.1 above and this section 4.2, it demonstrates the steps taken by our team to detect and track the ball. These tasks will be repeated until the end of the clip.

Sometimes, the tracking process may result in incorrect BoundingBox regions due to the continuous change of the ball's coordinates falling within the contour area of other places or being obscured by other points.

CHAPTER 5: RESULTS AND DISCUSSION

5.1. Evaluate Model

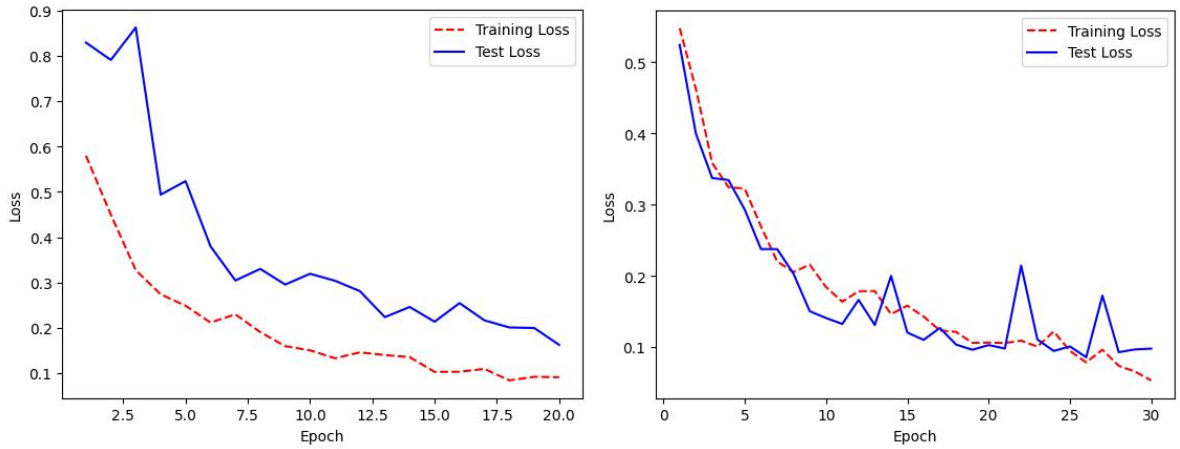


Figure 5.1: Train and loss history

Loss value: During the training process, it was observed that the value of loss decreased gradually from the first epoch to the last epoch. This indicates that the model is learning how to minimize errors.

Accuracy value: The accuracy of the training model increases gradually through each epoch, as seen from the graph. It can be observed that the model had an accuracy of only 50% in the first epoch and this increased steadily to 97% in the later epochs. This indicates that the model is learning how to classify the data samples accurately.

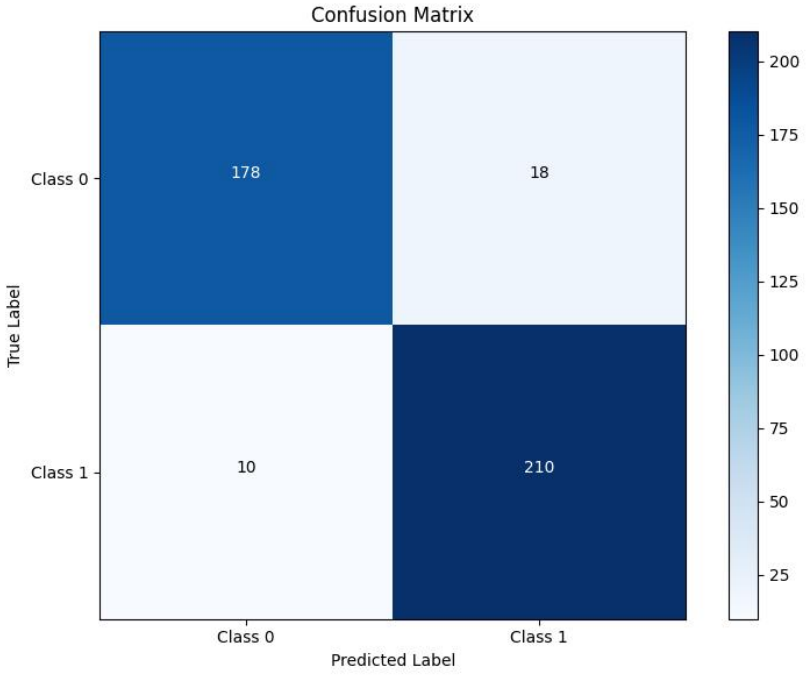
Validation Loss and Validation Accuracy value: The value of validation loss decreased gradually from the first epoch to the last epoch, from 0.5244 to 0.0978, during the training process. The value of validation accuracy increased from 0.8757 to 0.9661. This indicates that the model has good performance in prediction and is capable of accurately predicting the target values in the test file.

Other rating parameters:

Test Loss	**11%**
Test Accuracy	**93%**
Precision	**92%**
Recall	**95%**
F1-score	**93%**

Confusion Matrix:

With this result, we can identify that for Class 0, there were 178 true positive cases and 18 false positive cases, similarly for Class 1, there were 210 true positive



cases and 10 false positive cases. Overall, the model has high accuracy and reliability as the false positive cases were due to pre-arrangement in the test data file (section 2.2).

5.2. Discussion

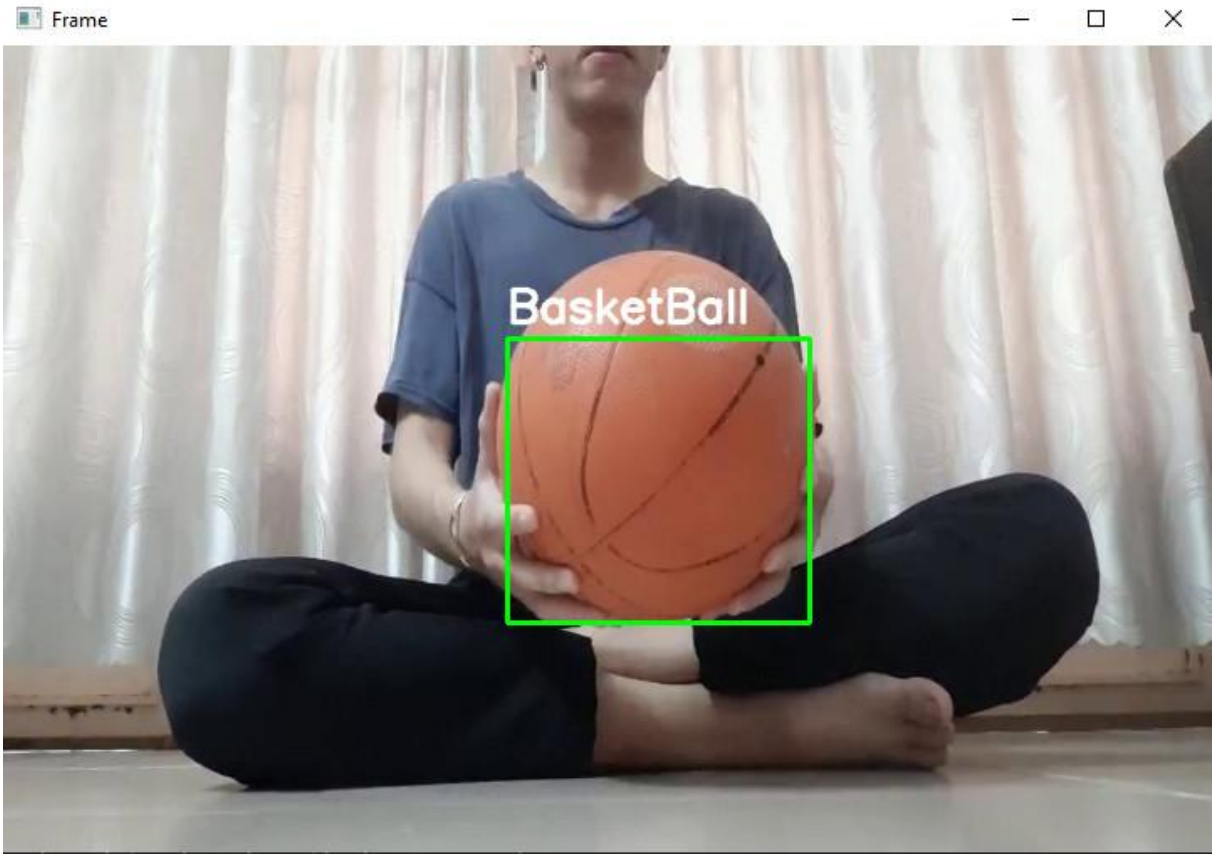


Figure 5.3: Result of detection and tracking ball

Through many experiments, to get better effects such as more accurate ball detection, less confusing ball tracking, we need to have a relatively large data source in the "non-ball" section. . Here the case of "not a ball" is quite small and the shooting angle does not change much, so it leads to many cases of detecting other orange objects such as wooden tables and chairs as basketballs. Besides, it is necessary to have a deeper understanding of the tracking algorithm so that the tracking of limited point coordinates is confused through another object.

In short, we also try very hard to find a solution on how to track so that we don't slow down the clip's time and get confused in a few cases, but it doesn't have much impact on the problem. This issue is completely resolved.

Finally, getting relative results also helps us better understand the subject of Machine Vision and its application in life. This subject is really interesting, it is not

easy to approach in a short time, it takes long-term accumulation to improve the ability to handle algorithms.

-END-

REFERENCES

1. [MÌ ÚP] Đọc ảnh trong nhiều folder và chia train, test siêu nhanh!, Mì AI, https://www.youtube.com/watch?v=zJR7bJHMMg0&ab_channel=M%C3%ACAI, 24/05/2023

2. Basket Ball Shot Predictor, CVZone, https://youtu.be/PVtmK8LmYpA, 23/05/2023