Pill Image Classification Software

K. Thaipakdee[1,2,*], P. Cen[1,2], M.Krachaechuang[1,2], S. Wiwatwitayawong[1,2], P. Ritthipravat[1]
[1]Department of Biomedical Engineering, Faculty of Engineering, Mahidol University, Nakhon Pathom, Thailand
[2]These authors contribute equally to this work. *thaipakdee.kha@gmail.com

***Abstract*— Nowadays, the pharmaceutical industry is making progress more effectively in order to manufactured medicines to cure disease. One major problem occurring to elders is taking the wrong medication. To solve this problem, the system that could help people identify the medication will be useful for public. Object detection technique is the selected computer vision technique because it can identify and locate objects in an image. Compared faster_rcnn_inception_v2_coco with ssd_inception_v2_coco, we decode to use Faster-RCNN as our model because has better rate of classification loss. The trained-model accuracy is 98.48% which can classify and locate pill in an image.**

Keywords—pill detection, object detection, machine learning, faster-rcnn-v2-coco,ssd-inception-v2-coco

I. INTRODUCTION

Nowadays, the pharmaceutical industry is making progress more effectively in order to manufactured medicines to cure disease. One major problem occurring to elders is taking the wrong medication. Factors that contribute wrong medication include poor eyesight and inability to read and memory problem. Despite these problems, many elders have to be responsible for following medication by themselves, while some of them may have their family members manage their activities. Moreover, it may be difficult for general public, not only elders, to see the differences of medicine by its appearance because many drugs look alike and are similar in size and colour. Many people who live alone with no one around to help them may find it harmful for medication error. To solve this problem, the system that could help people identify the medication will be useful for public.

In recent years, image processing has been used for pill recognition problem. The major features that are normally used are shape and colour of pill since many researchers found that using only colour would not be a good way to identify pills, thus, shape detection would also be used as a factor to determine pill identity. Machine learning is also another method of pill classification. The algorithm is to train samples to get the model that could be used for pill identification. Networks are used as the main classifiers to train the colour, shape, and characteristics of the pills, and the results are combined to obtain the final recognition results.

This project will provide this information as follows. Our 1st attempt to detect the drug by colour, shape and size, details of the created program, and results is discussed in section II. Section III presents the 2nd attempt which is mainly focused on machine learning with image pre-processing from data processing, object detection, machine learning training models, and results and discussion. The finished system will be easily used by computer software as shown in section IV. Finally, section V presents the discussion and conclusions, as well as possible ways to improve accuracy of the work.

II. FIRST ATTEMPT

We use colour and shape for detect and classify type of pill. Firstly, we equalize the images. After equalize image, then change the colour range from rbg to hsv and set threshold of colour value for each pill in different intensity of light source that the photo was taken by following code.

```
low_green = np.array([80, 10, 10])
high_green = np.array([120, 255, 255])
green_mask = cv2.inRange(hsv_image, low_green,
high_green)
green = cv2.bitwise_and(image, image,
mask=green_mask)
kernal = np.ones((3, 3), "uint8")
green_mask_di = cv2.dilate(green_mask, kernal)
```

Next, find position of pill where detect by colour and make a rectangle box around that area. Label that area with red letters.

```
contours, hierarchy =
cv2.findContours(green_mask_di,cv2.RETR_TREE,c
v2.CHAIN_APPROX_SIMPLE)
for pic, contour in enumerate(contours):
    area = cv2.contourArea(contour)
    if((area > 15000)and(area <30000)):
        x, y, w, h = cv2.boundingRect(contour)
       frame = cv2.rectangle(image, (x, y),(x + w, y +
h),(0, 255, 0), 2)
        cv2.putText(frame, "Green", (x,
y),cv2.FONT_HERSHEY_COMPLEX , 0.8,(0,0,
255))
```

From the result as Fig. 1. if the light source of pill image is changed, the threshold that setting for each colour will not match with colour in the image and finally cannot detect the pill. Green pill in second image is seem like black pill. So, we decide to use machine learning to detect the pill for improve the accuracy.

Fig. 1. First attempt results

III. SECOND ATTEMPT

To decide the computer vision techniques, firstly we need to understand user's problem. Elders always have many medicines to take in a course and when user take a photo of pill, they will take a photo of many pills together. So, we select object detection technique instead of image classification technique because object detection technique can identify and locate objects in an image.

A. Data Processing

1. Select 5 different pills that could be found in daily life to be classified in the project. Each of antibiotics, anti-allergy pill, antipruritic, and two of pain reliever pills are chosen. The selected pills and their properties are shown as Fig. 2.

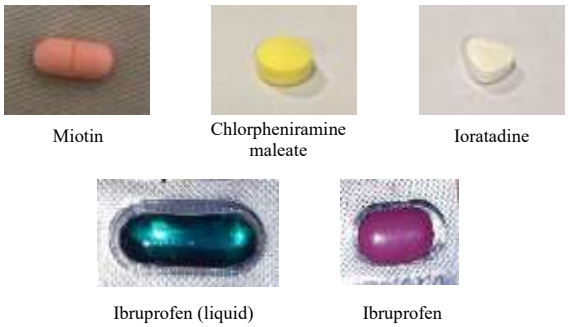


Fig. 2. Image of selected pills.

2. Take pictures of the fives pills. The pills must be taken in different positions in each image to use them to train the pill detector
3. Split picture for train dataset and test dataset in ratio 80:20. Then augment each group of datasets by modify contrast, brightness and gaussian blur.
4. Use labellmg to label 1000 images as id1 for an antipruritic, id2 for an anti-allergy pill, id3 for a green pain reliever, id4 for an antibiotic, and id5 for a purple pain reliever as Fig. 3. Labellmg saves a .xml files with the label data for each image.

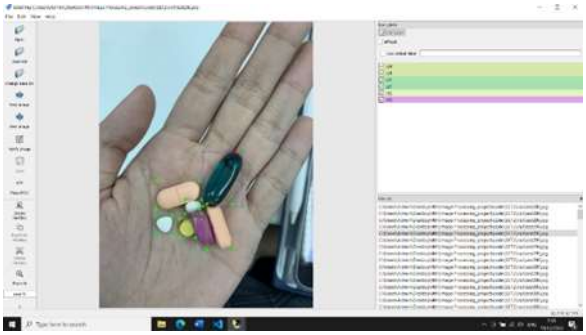


Fig. 3. Labelling image by Labellmg

B. Object Detection

Object detection is used in this project to localize and classify multiple pills in one image. The algorithm can define the position of the object in an image and draw a rectangle around it. The object in the box is then classify into 5 categories (id1 id2 id3 id4 id5) and is labelled with its type and the probability of that detection.

The TensorFlow Object Detection API is an open-source framework using TensorFlow to construct, train and deploy an object detection model. The source code can be cloned from https://github.com/tensorflow/models. Many pre-trained object detection models are also provided by TensorFlow at https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_ detection_zoo.md. The pre-trained model can be used to train with our own dataset, in this case pill images, instead of start training from scratch to reduce training time.

In this project, ssd_inception_v2_coco and faster_rcnn_inception_v2_coco are used as pre-trained models. SSD is a single-shot multi box detector which provides localization and classification at the same time while Faster-RCNN is a two-shot detector: localization and then detection. SSD is faster while Faster R-CNN is more accurate. Inception v2 is a convolutional neural network used for object detection. Before inception model is developed, the most popular CNNs is a stacked convolution layers by layers which may cause overfitting and is computationally expensive. The inception model uses multiple sizes filters (1x1, 3x3, 5x5) to operate on the same level then the outputs are concatenated and sent to the next inception module. In version 2, 5x5 convolution is changed into two 3x3 convolution operations to improve computational speed.

C. Methods

1. Install Anaconda
2. Create Anaconda Virtual Environment and activate

```
(base) C:\> conda create -n myenv pip python=3.7.1
(base) C:\> activate myenv
```

3. Install necessary packages

```
(myenv) C:\> python -m pip install --upgrade pip
(myenv) C:\> pip install --ignore-installed --upgrade tensorflow
(myenv) C:\> conda install -c anaconda protobuf
(myenv) C:\> pip install pillow
(myenv) C:\> pip install lxml
(myenv) C:\> pip install Cython
(myenv) C:\> pip install contextlib2
(myenv) C:\> pip install jupyter
(myenv) C:\> pip install matplotlib
(myenv) C:\> pip install pandas
(myenv) C:\> pip install opencv-python
```

4. Download TensorFlow Object Detection API repository from GitHub into folder project (Source: https://github.com/tensorflow/models)
5. Download the pre-trained model from TensorFlow's model zoo: ssd_inception_v2_coco and faster_rcnn_inception_v2_coco then extracts file to C:/project/models/research/object_detection (Source: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_ detection_zoo.md)
6. Set PYTHONPATH environment variable

```
C:\> set PYTHONPATH=C:\project\models;C:\project\models\research;C:\project\models\research\slim
```

7. Clone this repository into object_detection folder (Source: https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10#2-set-up-tensorflow-directory-and-anaconda-virtual-environment)
   Erase all files in \object_detection\images\train, \object_detection\images\test, \object_detection\training, \object_detection\inference_graph, "test_labels.csv" and "train_labels.csv" files in \object_detection\images
8. Compile Protobufs and run setup.py
   Protobuf files are used by TensorFlow to configure model and training parameters.

```
(myenv) C:\project\models\research> protoc --
python_out=.
.\object_detection\protos\anchor_generator.proto
.\object_detection\protos\argmax_matcher.proto
.\object_detection\protos\bipartite_matcher.proto
.\object_detection\protos\box_coder.proto
.\object_detection\protos\box_predictor.proto
.\object_detection\protos\eval.proto
.\object_detection\protos\faster_rcnn.proto
.\object_detection\protos\faster_rcnn_box_coder.proto
.\object_detection\protos\grid_anchor_generator.proto
.\object_detection\protos\hyperparams.proto
.\object_detection\protos\image_resizer.proto
.\object_detection\protos\input_reader.proto
.\object_detection\protos\losses.proto
.\object_detection\protos\matcher.proto
.\object_detection\protos\mean_stddev_box_coder.pro
to .\object_detection\protos\model.proto
.\object_detection\protos\optimizer.proto
.\object_detection\protos\pipeline.proto
.\object_detection\protos\post_processing.proto
.\object_detection\protos\preprocessor.proto
.\object_detection\protos\region_similarity_calculator.
proto
.\object_detection\protos\square_box_coder.proto
.\object_detection\protos\ssd.proto
.\object_detection\protos\ssd_anchor_generator.proto
.\object_detection\protos\string_int_label_map.proto
.\object_detection\protos\train.proto
.\object_detection\protos\keypoint_box_coder.proto
.\object_detection\protos\multiscale_anchor_generator
.proto .\object_detection\protos\graph_rewriter.proto
.\object_detection\protos\calibration.proto
.\object_detection\protos\flexible_grid_anchor_genera
tor.proto

(myenv) C:\project\models\research> python setup.py
build
(myenv) C:\project\models\research> python setup.py
```

9. Prepare data for training
   Move 1000 images with its label prepared in the data processing in to object_detection\images. 800 are used as train, other 200 are used as test. Change .xml label into .csv file by running

```
(myenv) C:\project\models\research\object_detection>
python xml_to_csv.py
```

   It will create a train_labels.csv and test_labels.csv file in the \object_detection\images folder.
10. Generate the TFRecord files
    Edit generate_tfrecord.py and replace our class into this part of code

```
# TO-DO replace this with label map
def class_text_to_int(row_label):
    if row_label == 'id1':
        return 1
    elif row_label == 'id2':
        return 2
    elif row_label == 'id3':
        return 3
    elif row_label == 'id4':
        return 4
    elif row_label == 'id5':
        return 5
    else:
        None
```

    Generate the TFRecord files by running

```
(myenv) C:\project\models\research\object_detection>
python generate_tfrecord.py --
csv_input=images\train_labels.csv --
image_dir=images\train --output_path=train.record
(myenv) C:\project\models\research\object_detection>
python generate_tfrecord.py --
csv_input=images\test_labels.csv --
image_dir=images\test --output_path=test.record
```

    It will create a train.record and a test.record file in \object_detection folder.
11. Create labelmap.pbtxt file in \object_detection\training folder.

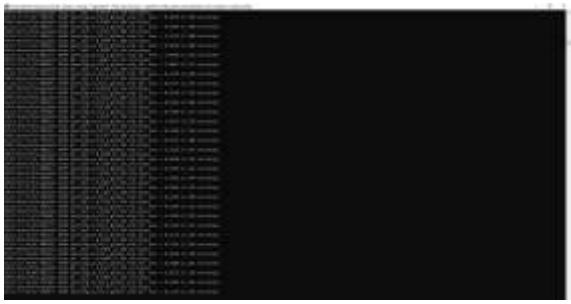
```
item {
  id: 1
  name: 'id1' }
item {
  id: 2
  name: 'id2' }
item {
  id: 3
  name: 'id3' }
item {
  id: 4
  name: 'id4' }
item {
  id: 5
  name: 'id5' }
```

12. Configure training
    Go to C:\project\models\research\object_detection\samples\configs and copy the selected model config file into \object_detection\training folder. Edit num_classes to 5. Edit input_path for train.record and test.record. Edit label_map_path for labelmap.pbtxt. Edit num_examples to 200 (number of test images)

13. Train the model

```
(myenv) C:\project\models\research\object_detection>
python train.py --logtostderr --train_dir=training/ --
pipeline_config_path=training/model_name.config
```

The result should be as follow



14. Use TensorBoard to look at the training process

```
(myenv) C:\project\models\research\object_detection>
tensorboard --logdir=training
```

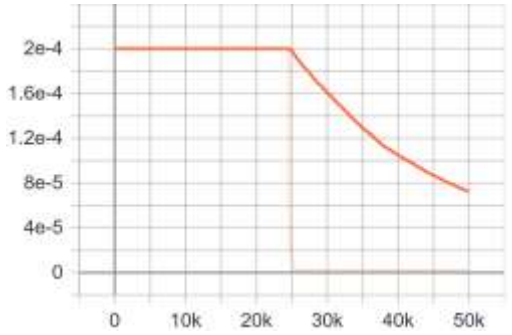
15. Generate the frozen inference graph (.pb file) after the training process is done.
Replace XXXX with the highest number of .ckpt file

```
(myenv) C:\project\models\research\object_detection>
python export_inference_graph.py --input_type
image_tensor --pipeline_config_path training/
model_name.config --trained_checkpoint_prefix
training/model.ckpt-XXXX --output_directory
inference_graph
```

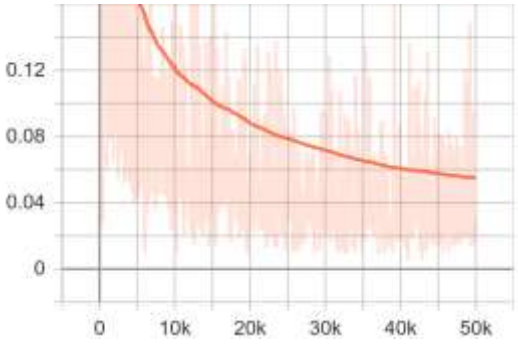
D. Results

1. faster_rcnn_inception_v2_coco

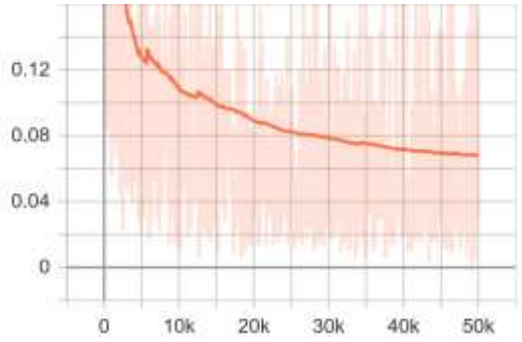
Learning Rate: The initial learning rate is set as 0.0002. 0.00002 for the first 10,000 steps. 0.000002 for the first 25,000 steps. 0.0000002 for the first 50,000 steps.



Localization Loss: Reduce after the number of steps increase. The final value is around 0.05 at 50,000 steps.

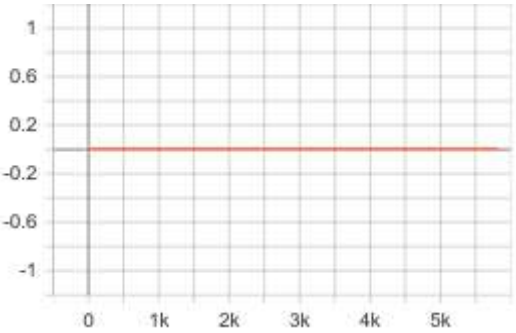


Classification Loss: Reduce after the number of steps increase. The final value is around 0.07 at 50,000 steps.

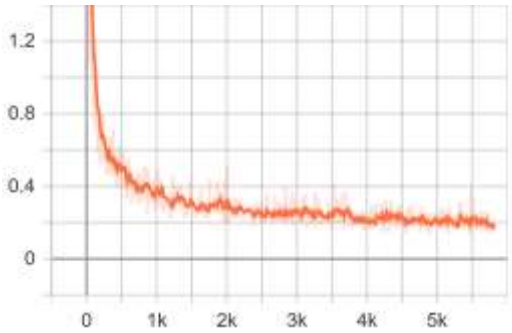


2. ssd_inception_v2_coco

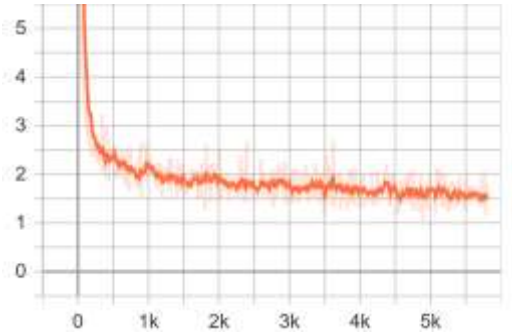
Learning Rate: The initial learning rate is set as 0.004



Localization Loss: Reduce after the number of steps increase. The final value is around 0.02 at 6,000 steps. The fluctuation is less than faster-RCNN model.



Classification Loss: Reduce after the number of steps increase. The final value is around 1.5 at 6,000 steps.



Due to low reduction rate of classification loss after long training time, we decided to terminate the training process and use Faster-RCNN as our model.

Model accuracy of Faster-RCNN

$$Accuracy\ (ACC) = \frac{\Sigma\ True\ positive\ +\ \Sigma\ True\ negative}{\Sigma\ Total\ population} = \frac{201 + 127 + 135 + 168 + 146}{789} = 0.9848$$

Confusion Matrix: 150 images were taken and used to validate the model accuracy and create the confusion matrix.

Predicted Class

Actual Class	class	id 1	id 2	id 3	id 4	id 5	unknown	Total
	id 1	201	0	0	0	0	2	203
	id 2	0	127	0	0	0	4	131
	id 3	0	0	135	0	0	0	135
	id 4	1	0	0	168	0	1	169
	id 5	0	0	0	0	146	0	146
	unknown	3	0	1	1	0	0	5
	Total	205	127	136	169	146	7	789

Example Result Image

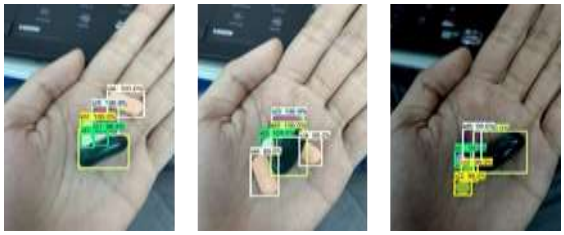


Fig. 4. Result Image

IV. GUI APPLICATION

1. Define trained model as function in python and its output is image

```
def load(image):
    # Name of the directory containing the object detection
module we're using
    MODEL_NAME = 'inference_graph'
    # Grab path to current working directory
    CWD_PATH = os.getcwd()
    PATH_TO_CKPT =
os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference
_graph.pb')
    PATH_TO_LABELS =
os.path.join(CWD_PATH,'training','labelmap.pbtxt')
    NUM_CLASSES = 5
    label_map =
label_map_util.load_labelmap(PATH_TO_LABELS)
    categories =
label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES,
use_display_name=True)
    category_index =
label_map_util.create_category_index(categories)
    detection_graph = tf.Graph()
    with detection_graph.as_default():
        od_graph_def = tf.GraphDef()
        with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
            serialized_graph = fid.read()
            od_graph_def.ParseFromString(serialized_graph)
            tf.import_graph_def(od_graph_def, name='')
    sess = tf.Session(graph=detection_graph)
    image_tensor =
detection_graph.get_tensor_by_name('image_tensor:0')
    detection_boxes =
detection_graph.get_tensor_by_name('detection_boxes:0')
    detection_scores =
detection_graph.get_tensor_by_name('detection_scores:0')
    detection_classes =
detection_graph.get_tensor_by_name('detection_classes:0')
    # Number of objects detected
    num_detections =
detection_graph.get_tensor_by_name('num_detections:0')

    image_rgb = cv2.cvtColor(image,
cv2.COLOR_BGR2RGB)
    image_expanded = np.expand_dims(image_rgb, axis=0)

    # Perform the actual detection by running the model with
the image as input
    (boxes, scores, classes, num) = sess.run(
        [detection_boxes, detection_scores, detection_classes,
num_detections],
        feed_dict={image_tensor: image_expanded})
    # Draw the results of the detection (aka 'visulaize the
results')
    vis_util.visualize_boxes_and_labels_on_image_array(
        image,
        np.squeeze(boxes),
        np.squeeze(classes).astype(np.int32),
        np.squeeze(scores),
        category_index,
        use_normalized_coordinates=True,
        line_thickness=6,
        min_score_thresh=0.9)
    return image
```

2. Develop GUI with Tkinter written in python makes user friendly ty browser image from their computer.

```
root = Tk()
root.title('Drag On Check')
root.configure(bg='azure3')
Header=Label(root, text="Drag On Check",
font=("Helvetica",35),bg='yellow2')
pick_btn=Button(root, text="Pick a photo",
command=open,font=("Helvetica",20),bg='salmon')
Header.grid(row=0, column=0)
#pad.grid(row=1,column=0)
#pick_btn.pack()
```

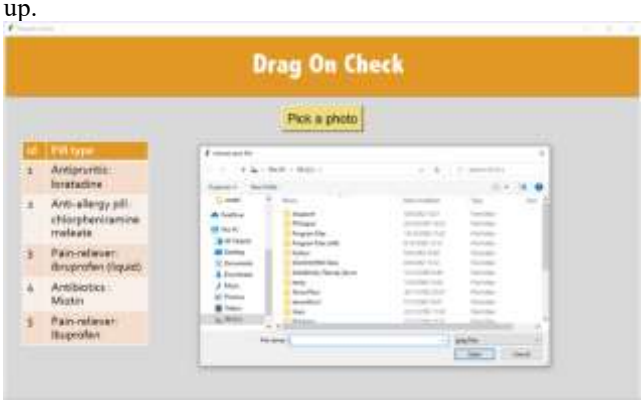
Show header and button. Click button to pick a photo.

3. Develop open function to directly browser image and input into our application.

```
def open():
    global my_image, my_rimage
    root.filename =  filedialog.askopenfilename(initialdir =
"C:/",title = "choose your file",filetypes = (("jpeg
files","*.jpg"),("all files","*.*")))
    #my_label=Label(root, text=root.filename).pack()

my_image=ImageTk.PhotoImage(Image.open(root.filename))
    print("root.filename")
    print(root.filename)
    path=root.filename
    img = cv2.imread(path)
```

After click Pick a photo button the file browser is show up.



4. Call function to detect and classify pill in function open.

```
def open():.
…
    ##Add object detection Model HERE ##
    print("function")
    resultimg=oad(img)
    ###################################
```

5. Show original and result image with label pill id and position.

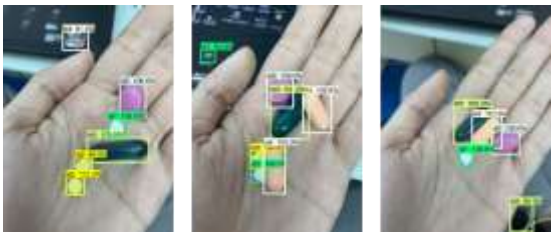
```
def open():
  …
    cv2.imwrite('Result.jpg', resultimg)
    Original=Label(root, text="Original")
    Original.grid(row=2, column=0)
    my_image_label=Label(image=my_image).grid(row=3,
column=0,padx=10,pady=10)
    OJD=Label(root, text="Drug detection")
    OJD.grid(row=2, column=1)

my_rimage=ImageTk.PhotoImage(Image.open("Result.jpg"))
    #ResultPhoto=PhotoImage(file="Result.jpg")
    my_rimage_label=Label(image=my_rimage).grid(row=3,
column=1,padx=10,pady=10)
…
```

V. DISCUSSION & CONCLUSION

From confusion matrix, there are false positive and false negative result images which lower the accuracy of the model.

Example Result Image (false positive)



(a) (b) (c)

Fig. 5. Result Image

The part of background which similar to the pill was detected by machine learning and conclude to be some type of pill which is incorrect because of nearly colour and shape. Solving these problems by using least distance from each pill and mostly position of pill in the image to classify that the part is background or pill with improve the accuracy.

Example Result Image (false negative)

(a) (b) (c)

Fig. 6. Result Image

From the result images indicate that the model cannot detect overlap pill because of its' less appear area in the image like image (a) and (b). Also, if the image shows different angle of most pill, the machine learning still cannot detect that pill like image (c) that normally orange pill will has long shape but in image (c), that pill looks more like circle pill.

We conclude that the model needs to improve the accuracy for having better result by doing more image orientation, increasing training data and reviewing COCO datasets. Doing image orientation will improve and increase possibility to detect more pill correctly in different angle of pill. Also, increasing training data will increase more accuracy of the model.

VI. REFERENCES

[1] A. (2020a, April 20). The Battle of Speed vs. Accuracy: Single-Shot vs Two-Shot Detection Meta-Architecture. Retrieved December 15, 2020, from https://allegro.ai/blog/the-battle-of-speed-accuracy-single-shot-vs-two-shot-detection/

[2] Bose, S. R., & Kumar, V. S. (2020). Efficient inception V2 based deep convolutional neural network for real-time hand action recognition. IET Image Processing, 14(4), 688–696. https://doi.org/10.1049/iet-ipr.2019.0985

[3] E. (2020b). EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10. Retrieved December 15, 2020, from https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10#2-set-up-tensorflow-directory-and-anaconda-virtual-environment

[4] Raj, B. (2020, July 31). A Simple Guide to the Versions of the Inception Network. Retrieved December 15, 2020, from https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

[5] Sharma, P. (2020, October 19). Computer Vision Tutorial: A Step-by-Step Introduction to Image Segmentation Techniques (Part 1). Retrieved December 15, 2020, from https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/