# COMPUTER VISION FINAL PROJECT

# BANANA DETECTION

**Aras Umut Erarslan**

## I. Introduction

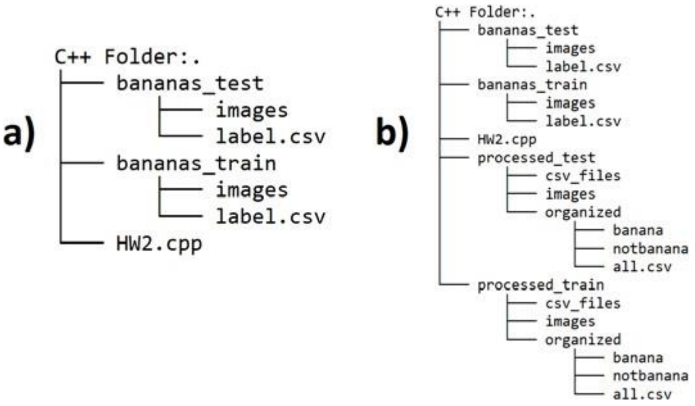
The banana detection project is developed by both C++ and Python. C++ is used in the pre-processing of the given dataset and converting it into a shape that will be used to train the model to detect bananas. Python is also used in a small part of pre-processing but it is mainly used while training the model and detecting bananas.

## II. Pre-processing

### II.I C++

The aim of the pre-processing part is, preparing the dataset to be trained. For this purpose, the given dataset is processed into 2 classes which are “banana” and “notbanana”. All the new images are cropped from the original images given in the dataset. After the cropping, the cropped images and their coordinates are compared with the labels of the given bananas in the images. If the image contains at least 256 banana pixels(1/4 of the whole cropped image), the cropped images are labeled as a banana which is class 1. Otherwise, cropped images are labeled as -1 and saved to the folder.

The C++ code saves the cropped images and their labels in 2 different ways and in the rest of the project only 1 way is used. The first way is saving every cropped image of an image in different folders and saving their labels in a csv format in different files(“csv_files” and “images” folders). The second way which is used in the Python code is, saving “banana” and “notbanana” images in separate folders and holding their labels in the same csv file(“organized” folder)(Fig.1.). As the result of C++ code, 3842 “banana” and 60158 “notbanana” images are cropped from the training dataset. Furthermore, 389 “banana” and 6011 “notbanana” images are cropped from the test dataset. The most important part of C++ code is, it requires to be chosen if the pre-processing will be done on train or test banana folders. It can be changed from the first line of the “main” function. The rest of the code and folder names will be changed depending on this choice.



```
C++ Folder:.
├── bananas_test
│       ├── images
│       └── label.csv
├── bananas_train
│       ├── images
│       └── label.csv
└── HW2.cpp
```

```
C++ Folder:.
├── bananas_test
│       ├── images
│       └── label.csv
├── bananas_train
│       ├── images
│       └── label.csv
├── HW2.cpp
├── processed_test
│       ├── csv_files
│       ├── images
│       └── organized
│               ├── banana
│               ├── notbanana
│               └── all.csv
└── processed_train
        ├── csv_files
        ├── images
        └── organized
                ├── banana
                ├── notbanana
                └── all.csv
```

*Fig.1. a) shows the overview of the folder tree before executing HW2.cpp for “test” and “train” data b) shows the overview of the folder tree after the execution of HW2.cpp for “test” and “train” data*

Here is the how pre-processing is done by C++ code step by step:

- Read labels from the csv file of the dataset.
- Create required folders.
- Read the images 256x256 images and crop them into 64 32x32 images.
- Compare the coordinates of 32x32 images and the csv file to check if any of the 32x32 images contains a part of a banana. The threshold is; if at least 1/4 of a cropped image(256 pixel) is banana, label the cropped image as “banana” which is “1”. Otherwise, label it as “notbanana” which is “-1”. Print the progress in the console.
- Save the cropped images and their labels. The target data is saved in 2 different ways. Only the “organized” folder is used in future processes. Images are names as “{Original Image Name}-{Crop Number}.png”. For example, if the name of a cropped image is 11-38, the cropped image is cropped from the 11th image and it is the 38th cropped image. Also in this way, all the coordinates of all cropped images are kept for future usage.

### II.II Python (pre-process.py)

Python is used to eliminate the cropped images and prepare the new dataset for the final form. As it is mentioned before, 3842 “banana” and 60158 “notbanana” images are cropped from the training dataset. After the cropping, banana images have some images without bananas. This situation caused by the annotating way of the original dataset which is bounding boxes. Bananas are not square or rectangle so there are some spaces without bananas in the bounding boxes. Some cropped images are intersected with these spaces and labeled as “banana” even they do not have even a pixel of a banana inside of them. Python code is used to eliminate these cropped images which do not have any banana pixel inside of them by checking the color of whole pixels in the image. After the tests, the threshold for the elimination is selected as 9. That means, cropped images which do not have at least 9 yellow pixels inside of them are eliminated and removed from the “banana” list. To check yellow pixels, RGB channels are analyzed 1 by 1 and checked separately. If a pixel satisfies the constraints below, it is considered a yellow pixel;

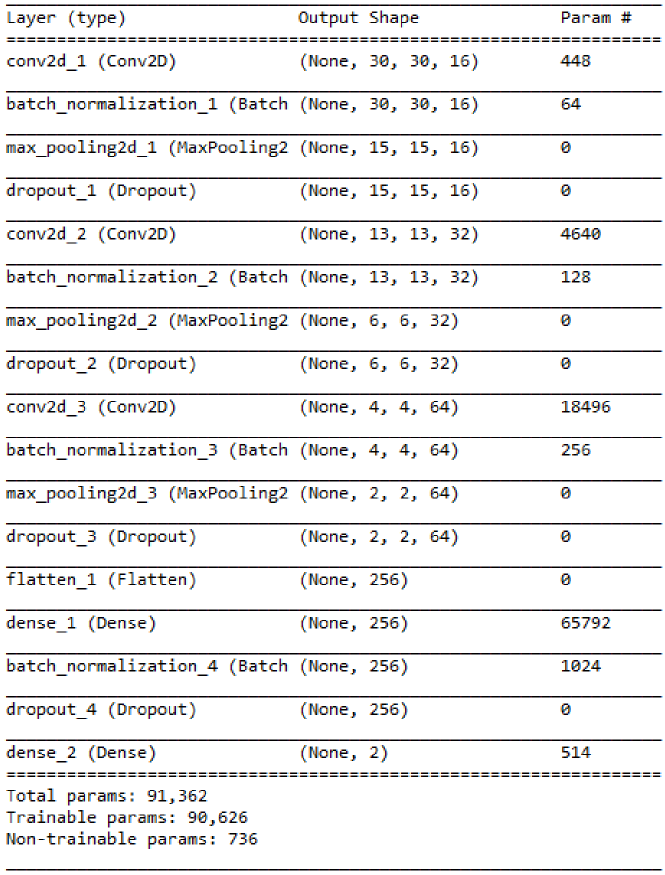
- $174 < R < 256$
- $174 < G < 256$
- $0 < B < 141$

As a result of the elimination, 3842 cropped “banana” images are dropped to 3207 images.

The training part requires a dataset with balanced classes. For this purpose, from the 60158 “notbanana” images, 56951 of them are randomly eliminated and a new dataset with 3207 “banana” and 3207 “notbanana” images is created under the processed_train -> eliminated folder with a new csv belongs to the created dataset.

## III. Creating Model (create-model.py)

The model with 3 Conv2d layers and between these layers, between these layers batch normalization, max pooling, and dropout is used. After flattening the model, 2 dense layers with batch normalization and dropout among them are used(Fig.2.). The models with more layers are tried but none of them gave better results. Early stopping with the patience of 10 is used and in every training try, the training is stopped by early stopping around epoch 20. A learning rate of 0.00001 is used. ImageDataGenerator function from Keras is used to create data generators for both training and validation steps. Parameters of the train data generator are selected as rotation_range=15, rescale=1./255, shear_range=0.1, zoom_range=0.2, horizontal_flip=True, width_shift_range=0.1, height_shift_range=0.1. Training data is split into train and validation datasets with the rate of 80/20% via using the train_test_split function from sklearn.model_selection. In the output layer, the activation function is used as “softmax” to be used in future calculations to understand which image is a better banana. The training epoch is selected as 200 but in all tries, training is stopped by an early stop. Model accuracy and validation accuracy is taken by model.history to be visulized epoch by epoch in a plot at the end. Batch size is taken as 15. At the end, the trained model is saved to be used in predict.py.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 30, 30, 16)        448       
_________________________________________________________________
batch_normalization_1 (Batch (None, 30, 30, 16)        64        
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 16)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 15, 16)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 32)        4640      
_________________________________________________________________
batch_normalization_2 (Batch (None, 13, 13, 32)        128       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 32)          0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 32)          0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 64)          18496     
_________________________________________________________________
batch_normalization_3 (Batch (None, 4, 4, 64)          256       
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 64)          0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 2, 2, 64)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792     
_________________________________________________________________
batch_normalization_4 (Batch (None, 256)               1024      
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 514       
=================================================================
Total params: 91,362
Trainable params: 90,626
Non-trainable params: 736
_________________________________________________________________
```

*Fig.2. model.summary() of the model*

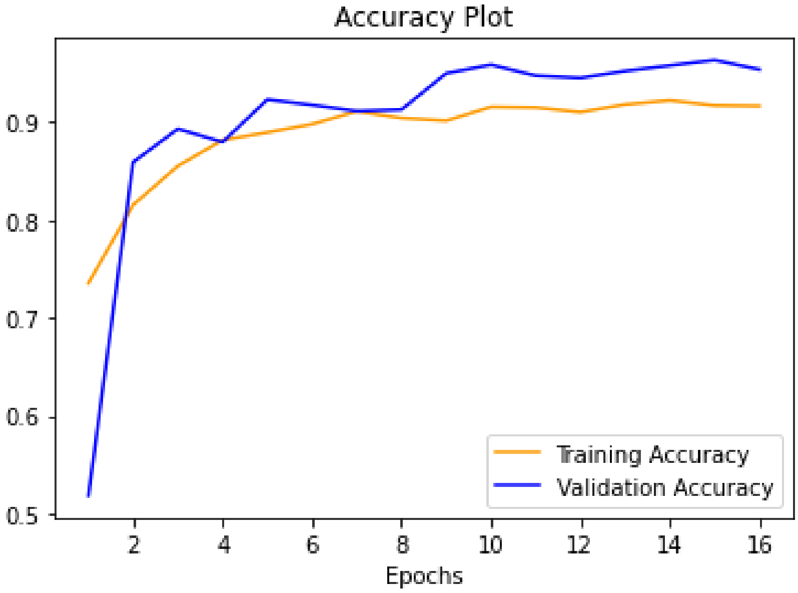


*Fig.3. training and validation accuracy of the model. (Early stop at 16th epoch)*

### IV. Prediction and Banana Detection (predict.py)

The code starts with a section that has 4 variables. These 4 variables can be changed to change the result of the banana detection. Image path is the path of 256x256 image that will be used to detect banana inside of it. About the other 3 variables, more details will be given later.

After loading the model, “generated_image” size of 32x32 images are randomly generated from the given 256x256 image. Later, prediction is done on these randomly generated images and if the randomly generated image is a “banana”, the yellow pixels in the image are counted(Fig.4.).

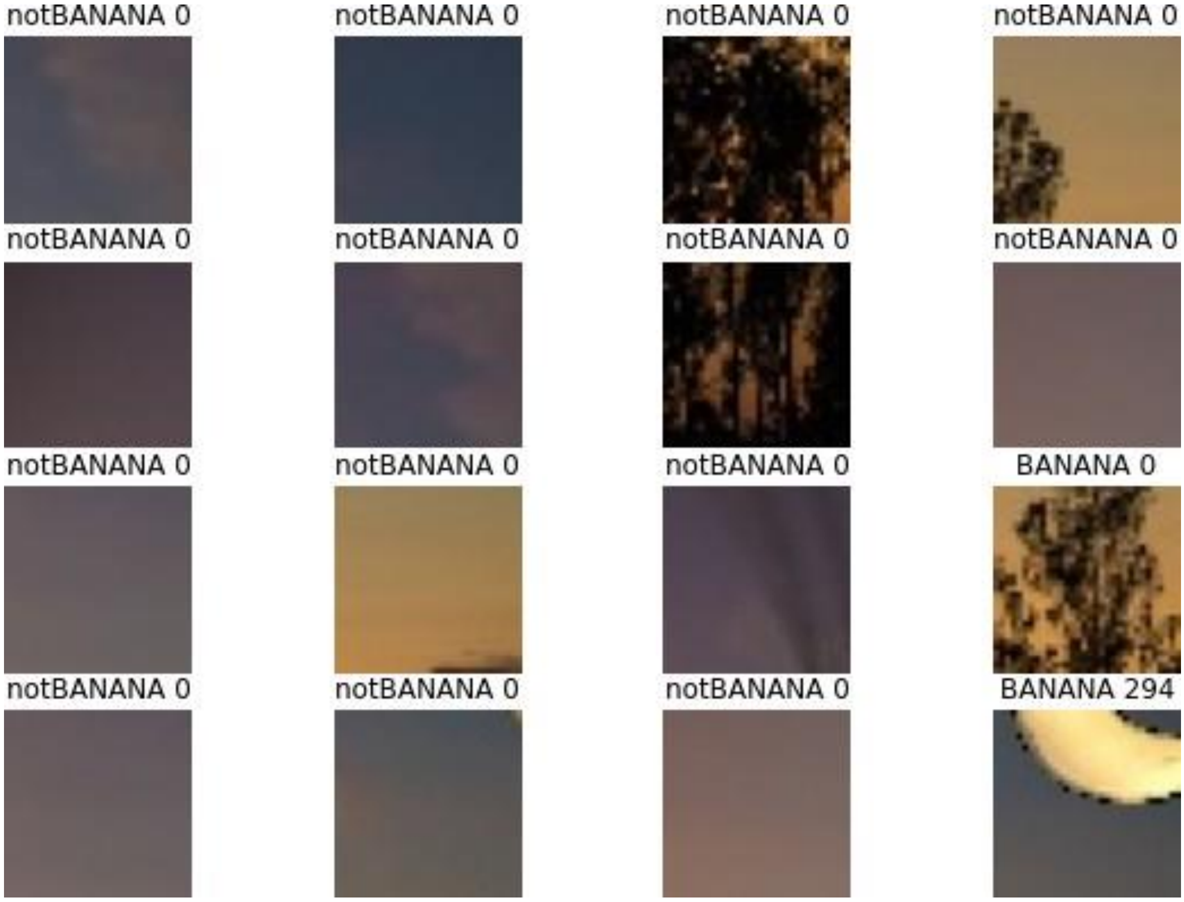


*Fig.4. the randomly generated images and the result of the prediction and the yellow pixel count above them. Even there is a bad prediction, yellow pixels are counted to get good results.*

The function to count yellow pixels is the same as the one used in the pre-process.py . This process is done to determine the best banana among the randomly generated bananas. At first, a banana score is considered to determine the best banana but after analyzing the results, it is determined to use yellow pixel detection to determine the best banana.

When the best banana is selected, its coordinates are used to generate new 32x32 images from the original image that is around the best banana(Fig.5.). Here the “distance” variable is used and it can be changed from the first section of the code to get different results of banana detection. “Distance” refers to the pixel count that determines how far a newly generated image can be from the best banana.

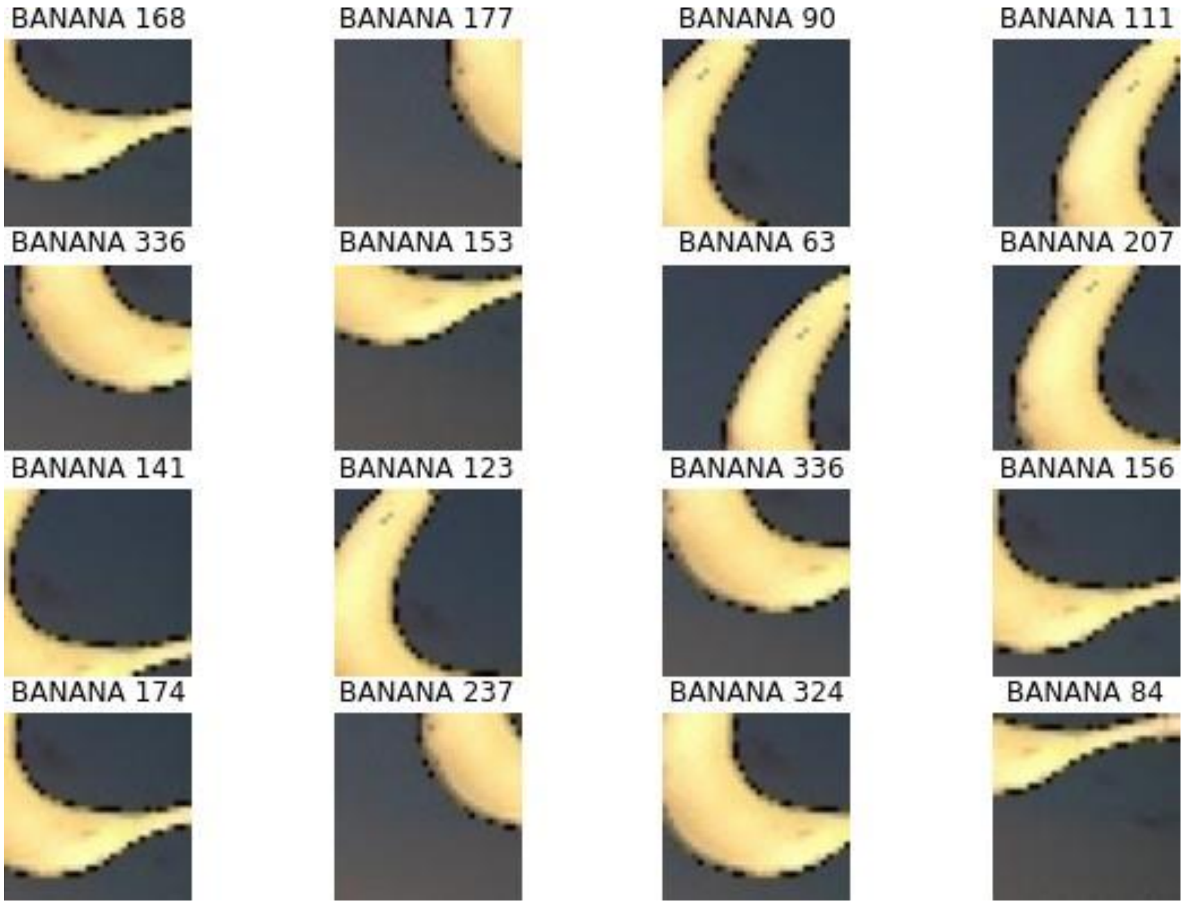


*Fig.5. the randomly generated images around the best banana using the “distance” variable to determine distance from the best banana.*

Later, “strict_check” variable from the first section is used to eliminate newly generated bananas. “strict_check” determines the threshold of yellow pixel counts in the images. For example, if the “strict_check” is 200, images that have yellow pixels below 200 are eliminated. After the eliminations, coordinates of the banana images are merged to create the bounding box of the banana. For this purpose, min(x), min(y), max(x), max(y) coordinates are used. Different strict_check and distance values can be used to get different results(Fig.6.). When distance is increased, decreasing strict_check gives better results most of the time.

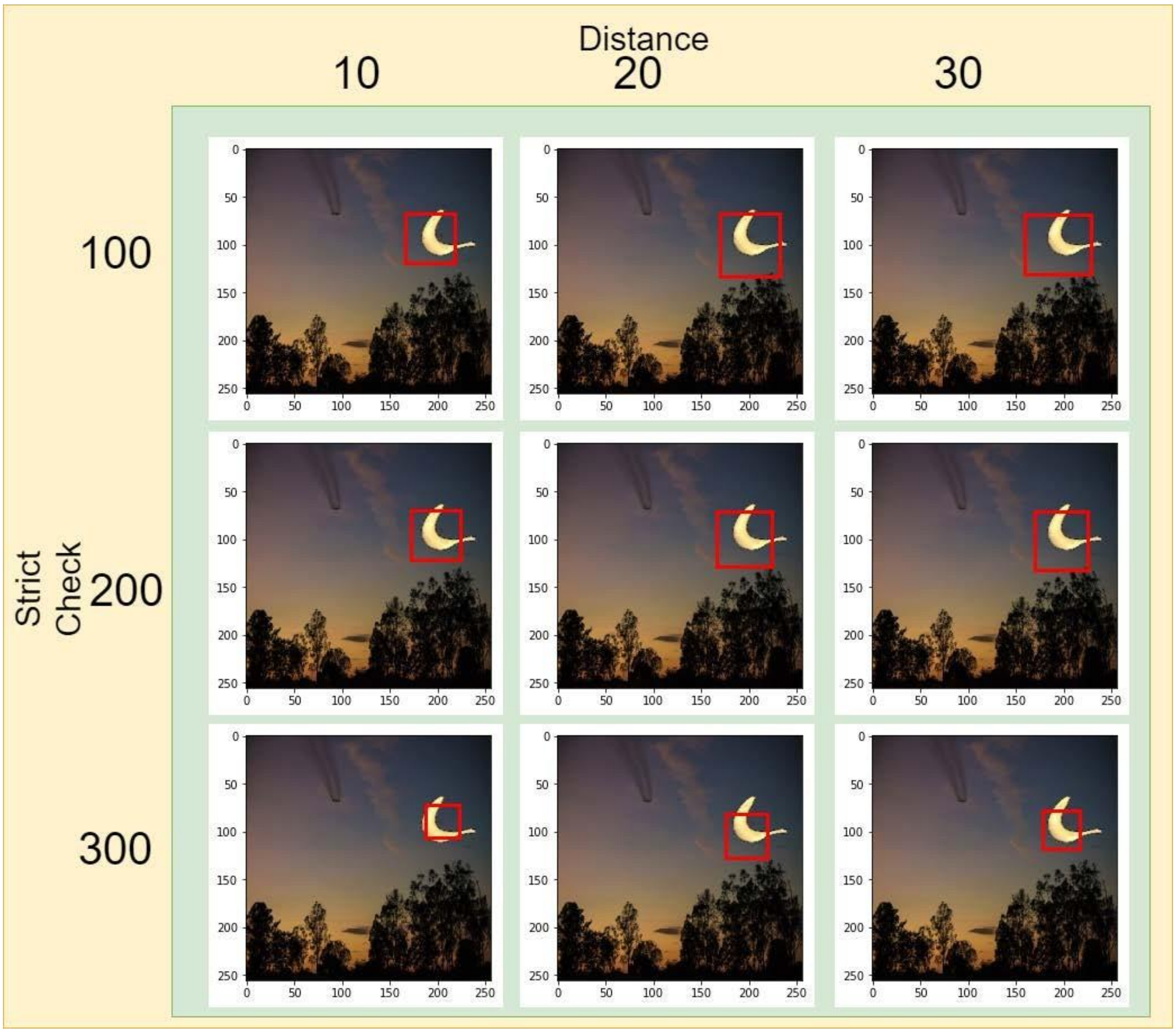


*Fig.6. "generated_image" is 100 in all examples in the figure. overview to the examples with different values. (Note: after the tests, the best parameters for this dataset are found as 1000 for "generated_image", 10 -17 for "distance and" 200-400 for "strict_check")*

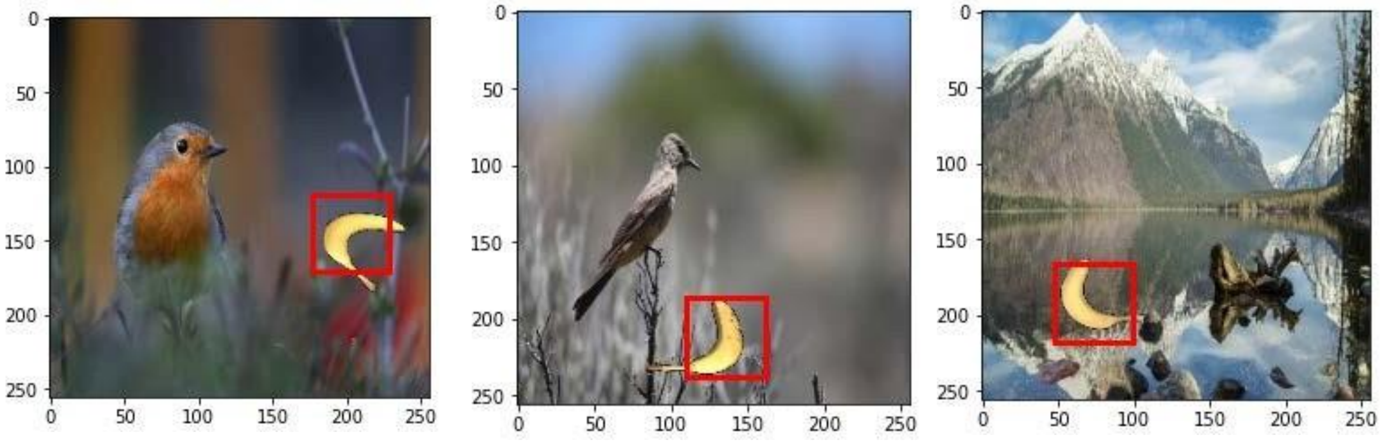


*Fig.7. banana detection with generated_image = 1000, distance = 10, strict_check = 200*

## V. Conclusion

Even though results seem good with distance of 10 and strict check of 200, fig.7 results require 1000 “generated images”. Generating 1000 images from the original image and generating another 1000 images around the best banana requires so much time (around 30-40 seconds). The algorithm can be improved by adding more steps to the elimination or writing elimination/generation steps with more layers to generate better images. Also, improving model accuracy increases the success of the banana detection drastically and this can be done by fine-tuning models like Yolo.