

**Human Vs Horse Classification & Object Tracking**

|  |  |  |
| --- | --- | --- |
| f |  |  |

**Faculty of Computers & Benha University**

**Artificial Intelligence**

**Horse Vs Human Classification**

**& Object Tracking**

**In**

Neural Network

**Team Members:**

* Ebrahem Amr Ebrahem Amin (sec 1)
* Ahmed Nasser Mahmoud Ahmed (sec 1)
* Kerolos Edwar Boshra (sec 3)
* Mohamed Ashraf Mohamed Shaban (sec 3)
* Mohamed Ghandour Mohamed (sec 4)

***Supervised by***

***Eng. Hisham Gamil***

***Dr. Hala Zayed***

**June 2021**

Table of Contents

[***1.*** **Human vs Horse Classification** 3](#_Toc76057193)

[***1.1*** ***Project Description*** 3](#_Toc76057194)

[***1.2*** ***Model we used*** 4](#_Toc76057195)

[***1.3*** ***Results*** 5](#_Toc76057196)

[***2.*** **Object Detection and Tracking** 8](#_Toc76057197)

[***2.1***  ***Project Description*** 8](#_Toc76057198)

[***2.1***  ***Model we used*** 8](#_Toc76057199)

[***2.2*** ***Flowchart*** 10](#_Toc76057200)

[***2.3*** ***Results*** 11](#_Toc76057201)

[**3** **Conclusion** 12](#_Toc76057202)

# **Human vs Horse Classification**

## ***Project Description***

This project aims to classify images and tell you if a given image contains a horse or a human. The model uses a dataset containing rendered images divided into two classes Horse & Human. The network is trained to recognize features that determine which is human or horse.

The dataset we use is from laurencemoroney.com. The dataset contains 500 rendered images of various species of horse in multiple poses in multiple locations. It also includes 527 rendered images of humans in different poses and backgrounds. The dataset also contains images which diverse in humans, so there are both men and women and Asian, Black, South Asian, and Caucasians present in the training set. The validation set adds six different figures to ensure breadth of data. As shown in horse figure 1, 2 and human figure 1, 2  
<https://laurencemoroney.com/datasets.html#horses-or-humans-dataset>

****<https://storage.googleapis.com/laurencemoroney-blog.appspot.com/horse-or-human.zip>

|  |  |  |  |
| --- | --- | --- | --- |
| **Horse Figure 1** | **Human Figure 1** | **Horse Figure 2** | **Human Figure 2** |

## ***Model we used***

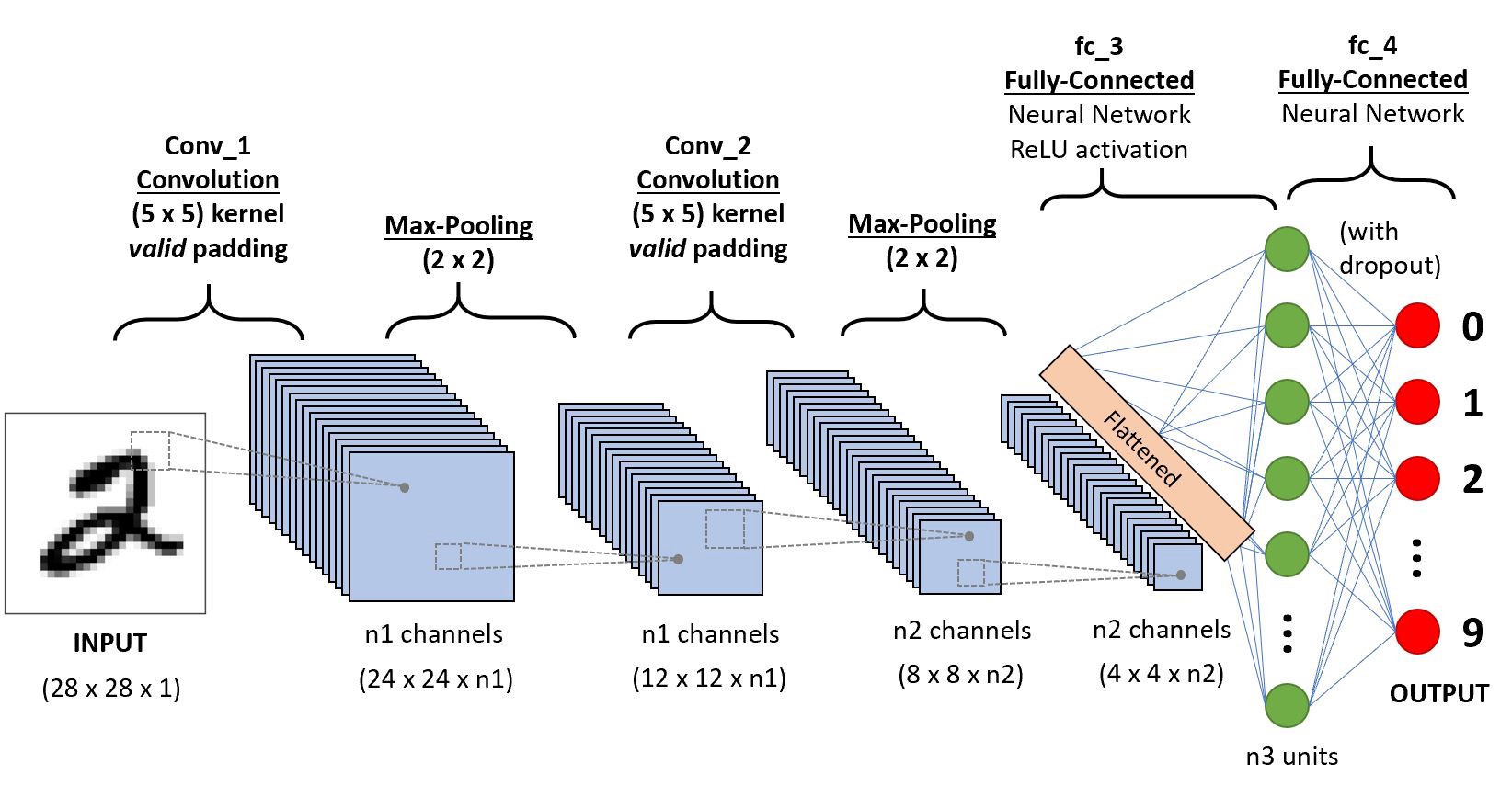
 We used Convolutional Neural Networks Model. A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The following figure represents the layers of CNN as shown in the following figure

Figure 1

We used Tensorflow framework with keras to build The CNN Network

* First we applied 16 filters of size 3x3 by applying RELU Activation function

model = tf.keras.models.Sequential([

tf.keras.layers.Conv2D(16, (3,3), activation='relu', input\_shape=(300, 300, 3)), tf.keras.layers.MaxPooling2D(2, 2),

* Then we applied 32 filters of size 3x3

tf.keras.layers.Conv2D(32, (3,3), activation='relu'),

    tf.keras.layers.MaxPooling2D(2,2),

* After that we build three layers each layer has 64 filters of size 3x3 and then applied to the image

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

* We used the maxpooling layer filter of size 2x2 along with the convolutional layers.

tf.keras.layers.MaxPooling2D(2,2),

* Finally we flattened the image and create 512 nodes to build the neural network and used the sigmoid activation function.

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(512, activation='relu'),

    tf.keras.layers.Dense(1, activation='sigmoid')])

**Optimizers:**

We used the RMSProp with learning rate 0.001

loss='binary\_crossentropy',optimizer=RMSprop(learning\_rate=0.001), =['acc']

## ***Results***

Results are shown in the following Table and Figures

Table 1

|  |  |  |  |
| --- | --- | --- | --- |
| Epoch No. | Time and Steps | Loss | Accuracy |
| 1 | 100s 12s/step | 0.7650 | 0.5651 |
| 2 | 96s 11s/step | 0.7231 | 0.6263 |
| 3 | 96s 11s/step | 0.9598 | 0.8042 |
| 4 | 97s 11s/step | 0.2762 | 0.8799 |
| 5 | 100s 12s/step | 0.3332 | 0.8854 |
| 6 | 97s 11s/step | 0.1851 | 0.9277 |
| 7 | 97s 11s/step | 0.1387 | 0.9477 |
| 8 | 95s 11s/step | 0.1233 | 0.9566 |
| 9 | 97s 12s/step | 0.5211 | 0.8710 |
| 10 | 98s 12s/step | 0.1968 | 0.9555 |
| 11 | 97s 11s/step | 0.0758 | 0.9722 |
| 12 | 94s 11s/step | 0.0403 | 0.9900 |
| 13 | 96s 11s/step | 0.4655 | 0.8721 |
| 14 | 103s 12s/step | 0.1016 | 0.9700 |
| 15 | 97s 11s/step | 0.0335 | 0.9911 |

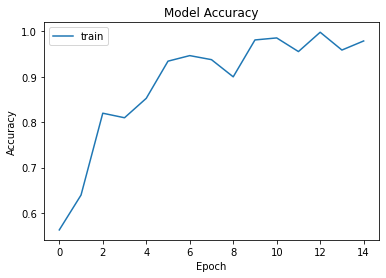
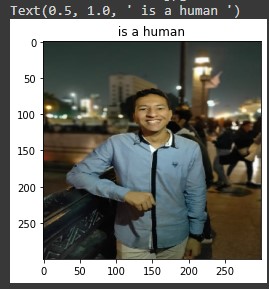
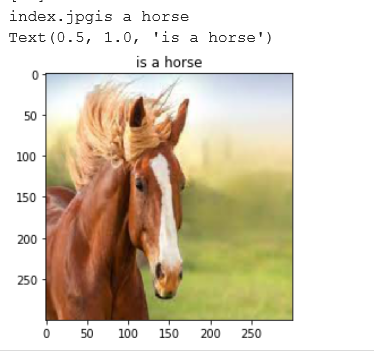
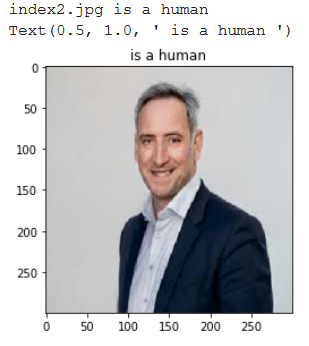
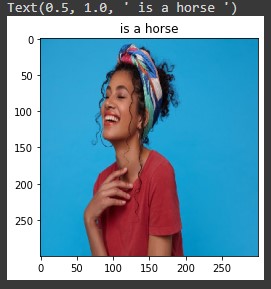


Figure 5 (Image of Human)

Figure 4 (Image of Human)

Figure 3 (Image of Horse)

Figure 2 (Model Accuracy on 15 Epochs)

****There was a little error in little images as shown in the following images

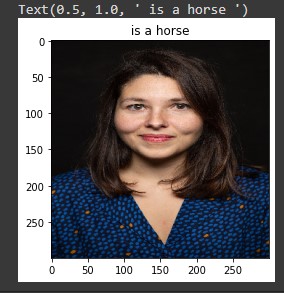
****

Figure 8 (Error at Image of Human)

Figure 7 (Error at Image of Human)

Figure 6 (Error at Image of Human)

# **Object Detection and Tracking**

## ***2.1 Project Description***

This project aims to detect objects from video and track objects within this video. We first choose the area that we want to detect then the model will detect objects in the area we have chosen. The object is determined by a yellow rectangle that appears on the video.

## ***2.1 Model we used***

* We used OpenCV to read video and frames

**def** ini(vidoe):  
 ret, frame = vidoe.read()  
 frame = cv2.resize(frame, (1280, 720))  
 imshow(**'Frame'**, frame)  
 roi = selectROI(**'Frame'**, frame)  
 destroyWindow(**'Frame'**)  
 **return** roi  
  
  
tracker = EuclideanDistTracker()  
video = VideoCapture(**'highway.mp4'**)

* After reading Video and Frames we choose area that we want to detect objects within this area. Then we press Enter button from keyboard to begin tracking.
* Then the selected area will be separated and viewed outside the main window (In RGB Mode and Black &White Mode)

object\_detector = createBackgroundSubtractorMOG2(history=100, varThreshold=40)

* Finally the objects are tracked with a yellow rectangle representing each object in the video
* Tracker Code

**def** update(self, objects\_rect):  
 *# Objects boxes and ids* objects\_bbs\_ids = []  
 *# Get center point of new object* **for** rect **in** objects\_rect:  
 x, y, w, h = rect  
 cx = (x + x + w) // 2  
 cy = (y + y + h) // 2  
 *# Find out if that object was detected already* same\_object\_detected = **False  
 for** id, pt **in** self.center\_points.items():  
 dist = math.hypot(cx - pt[0], cy - pt[1])  
 **if** dist < 25:  
 self.center\_points[id] = (cx, cy)  
 *# print(self.center\_points)* objects\_bbs\_ids.append([x, y, w, h, id])  
 same\_object\_detected = **True  
 break** *# New object is detected we assign the ID to that object* **if** same\_object\_detected **is False**:  
 self.center\_points[self.id\_count] = (cx, cy)  
 objects\_bbs\_ids.append([x, y, w, h, self.id\_count])  
 self.id\_count += 1

* Showing Images

boxes\_ids = tracker.update(detections)  
**for** box\_id **in** boxes\_ids:  
 x, y, w, h, id = box\_id  
  
 cv2.putText(roi, str(id), (x, y - 15), cv2.FONT\_HERSHEY\_PLAIN, 1, (255, 0, 255), 2)  
 cv2.rectangle(roi, (x, y), (x + w, y + h), (0, 255, 255), 1)  
  
imshow(**"frams"**, frams)  
imshow(**"maskafter"**, mask)  
imshow(**"msss"**, roi)  
key = waitKey(30)  
**if** key == ord(**'q'**):  
 **break**

## ***Flowchart***

No

If Area of Object > 1000

Start tracking the object

Adding Threshold to mask

Performing Subtracting Algorithm to detect Object

Yes

Yes

No

Pressing Enter from Keyboard

Drawing Contours on the object

Selecting Area to detect object

The Video is loaded

## ***Results***

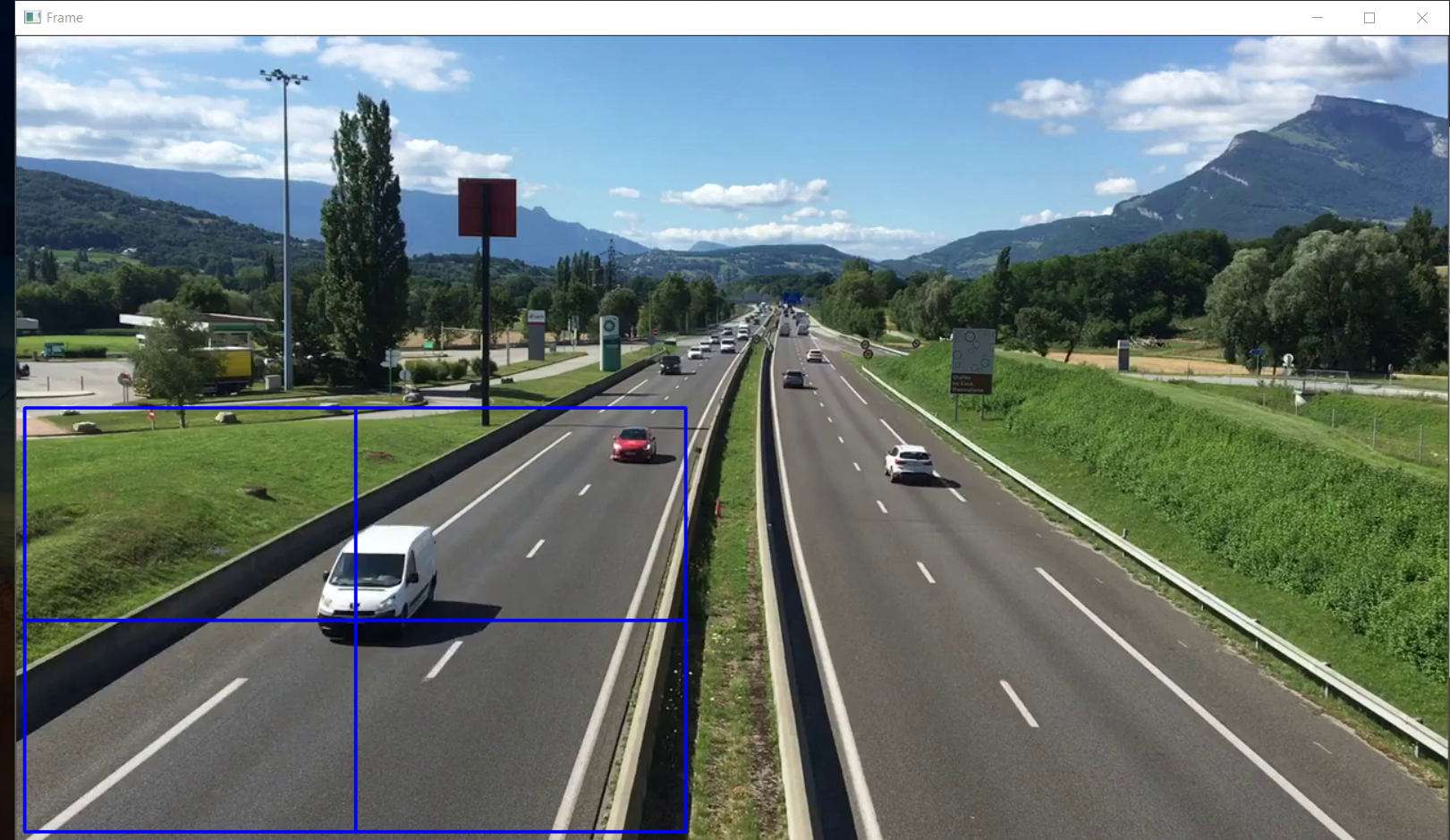
Results are shown in the Following Figures

Figure 10 (Running Project)

Figure 9 (Selecting ROI: Region of Interest)

# 

Figure 11 (Running Project)

# **Conclusion**

For Horse and Human Classification we applied CNN Model and the accuracy reached about 98% but with some error about detecting the images. The Accuracy is good at detecting human when image contains the upper body of human. For Object Detection and Tracking, the tracking algorithm we used is createBackgroundSubtractorMOG2 which subtracts the object from the main frame or image and it was quite good. This model is useful for beginners who want to learn tracking but there are many powerful algorithms that are good at tracking such as

* + SCRT
  + Median Flow Tracker
  + MOSSE