Multi-Object Detection and Tracking

Troy Daniels

11/09/2023

Contents:

Abstract -

Multi-object detection and tracking tasks using machine learning have become very successful today. This research is focused on understanding and implementing a relatively gentle but effective approach toward multi-object tracking by combining concepts from a number of different sources. Furthermore, it looks to extend the success of these concepts by creating new convolutional neural network architectures, changing certain learning tasks and other modifications such as replacing certain metrics. Although there are many other techniques used in multi-object tracking not mentioned here, the paper hopes to build a good foundation for understanding the general problem, obstacles that hinder its success and possible solutions to improve the technology.

I would like to thank Professor Michael Eckmann for all of his help throughout this project. His commitment to helping students inside and outside of the classroom is remarkable. I am so grateful for his support, knowledge, insight and patience!

Introduction -

Multi-Object Tracking (MOT) is the problem in which a computer is able to identify objects in a photo and track their movement throughout frames. Examples of these objects could be humans, cars, planes or just about anything else. MOT's tasks are generally to detect correct objects in a photo, locate the objects within the image, extract features from each object, maintain their identities based on features and compute trajectories by connecting objects with the same identities though several frames. MOT plays a crucial part in things such as autonomous driving, theft prevention, sports analytics and much more.

Today, MOT approaches using Computer Vision and deep learning have been far more successful than any other approach. Computer Vision is a subfield of computer science by which a computer is able to parse out information from a digital image. Deep learning, in this case, is using very large convolutional neural networks to process data in order to learn and use features from that data (figure 1). The idea of neural networks were derived from the structure of the human brain. In the human brain connections between neurons are strengthened when used. A similar idea is used here where weights are increased when they contribute to a correct output. This approach requires a convolutional neural network to be used at almost every part of the problem and sometimes even multiple networks for a single part. There are generally four parts or stages that MOT can be broken into.

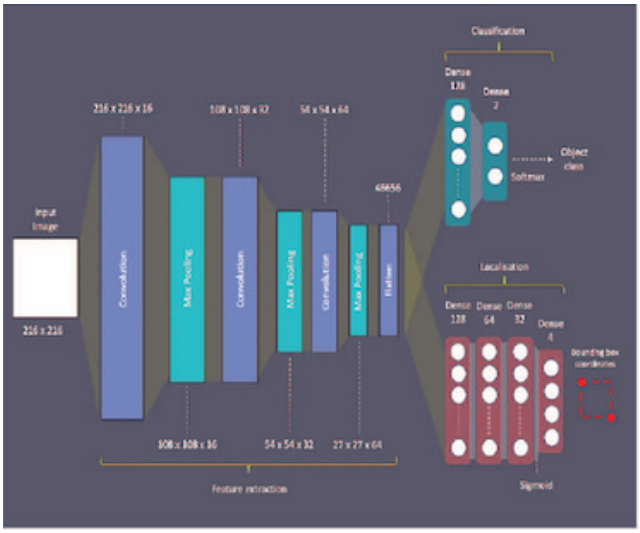


Figure 1.

The first stage is the detection stage. During this stage each specified object or objects should be detected in an image. Furthermore, the object's location in relation to the image needs to be learned (figure 2). Depending whether MOT is trying to detect different types of objects, classification, a task to identify an object's type from a predetermined set of classes, may also be used. Often the detection and localization are treated as one, since each localization of an object can be treated as a detection. This will be discussed in more detail later on.

Figure 2.

Next, is the feature extraction stage. This is where each object has features unique to itself parsed out and saved. For example, two red cars have more similar features than a red car and blue truck. The features extracted at this stage should be unique enough to be able to distinguish differences between objects (figure 3). There are several approaches to accomplish this stage, such as machine learning, segmentation, and histogram matching.

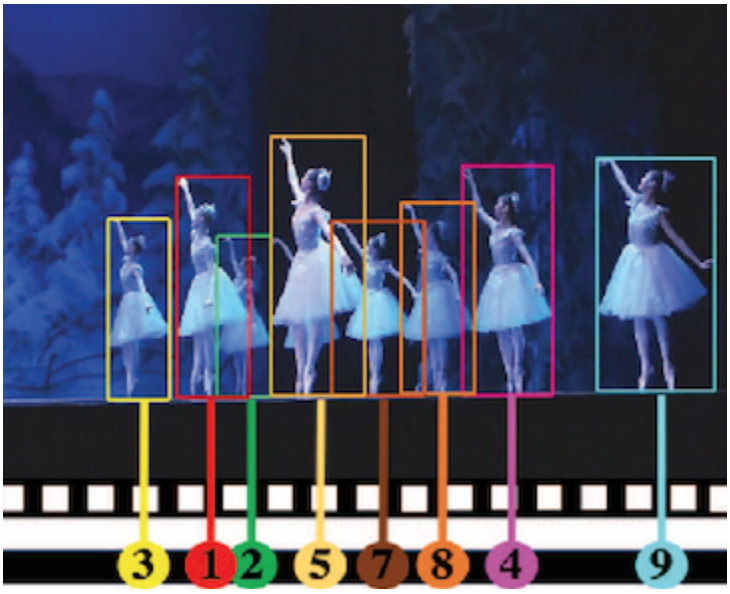


Figure 3.

The third is the affinity stage. Using the outputs of the previous two stages on many images, detected objects in different frames are grouped together based on a similarity score. This similarity score is based on localization (distance between two objects) and the features extracted (during the extraction stage) of two objects in adjacent frames. Two objects with a high similarity score are grouped together.

The last stage is the association stage. Here, each grouping obtained from the affinity stage is treated as one individual object moving through frames. The localization from the first stage is then used to compute trajectories or determine paths of each individual object (figure 4). The data learned from this stage can be used for further insights such as possible collisions (self-driving cars) or motives based on movement but these insights extend past the scope of this research.

Figure 4.

There are many other approaches to these stages that may differ from what was explained here based on the desired results of MOT. The major distinctions of MOT are whether the processing is in real time and if there are separate classifications that need to be detected and tracked. For example, live security footage vs. past sports footage and purely human tracking vs. human and automobile tracking. Real time processing is referred to as online, where using pre-collected data (like past sports footage) is called offline.

Bounding Box Regression -

The purpose of this research is focused on an offline MOT for tracking people. MOT starts with a fundamental need to detect and locate an object in a photo. To accomplish this, bounding box regression is used. Given an image of a person, the goal is to have a deep convolutional neural network locate that person in the image (detecting it by locating it). To do this, the output layer consists of four outputs which represent the coordinates (pixel coordinates) of an area that tightly captures the object (i.e. top left and bottom right of a bounding box) in question (figure 5). An example of the four outputs is as follows:

X1 = top left X coordinate of bounding box

Y1 = top left Y coordinate of bounding box

X2 = bottom right X coordinate of bounding box

Y2 = bottom right Y coordinate of bounding box

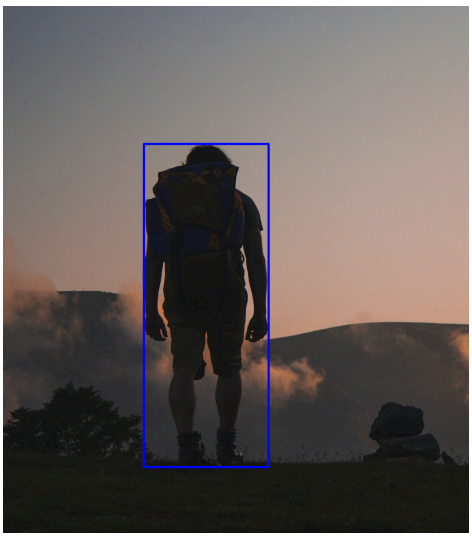


Figure 5.

In order to train a model for bounding box regression, a large data set of images of people (one person per image) and ground truth bounding box coordinates (hand selected X1,X2,Y1,Y2 coordinates for each image) are needed. The data also needs to be preprocessed. Each image needs to be resized so all are a consistent size. After doing so, the ground truth bounding box coordinates no longer correctly map to the object. Although the ground truth bounding box coordinates can be converted to match the new size of the image, it is preferable to normalize them in the range of (0,1). This range is preferable because we wish to avoid error calculations being influenced by the relative size of the bounding boxes compared to the entire image. Say we have a large predicted and ground truth bounding box that intersect for all but 10 pixels. We would not want smaller bounding boxes that also intersect for all but 10 pixels to have the same error as the former. As a result, normalized coordinates are computed by dividing each coordinate by the shape of the original image.

$X1 = X1 / W$

$Y1 = Y1 / H$

$X2 = X2 / W$

$Y2 = Y2 / H$

where:

W is the original width of the image

H is the original height of the image

Additionally, the pixel values should be normalized in order to reduce the mathematical complexity for the network which should result in faster and better results. The mapping of 0 (original pixel value) to -1 and 255 (original pixel value) to 1 is the most common.

For each pixel color channel (P) in the image:

$$P = (P - 127.5) / 127.5$$

After preprocessing each image and ground truth bounding box, the dataset is divided up into training, validation and test sets. The convolutional neural network's output layer consists of four outputs from a dense layer. Mean squared error or binary cross entropy can be used as the loss function but gives little indication of how well the model can predict a bounding box. Intersection over union (IoU) is a better metric to be used as the accuracy of the predicted bounding box. After training, if successful, given an image from the test set, the model should be able to predict the normalized coordinates of a bounding box.

Multi-Object Detection -

Bounding box regression is only capable of producing a single bounding box per image, which does not accomplish the task of multi-object detection. We need to generalize the task to detect and locate an arbitrary number of people in an image. To accomplish this, one approach is to train a bounding box regression model to work well on images where the object is located in the center. Then break the original photo in segments by sliding a frame (of a predetermined size) across the entire photo. Run each segment in the bounding box regression model to produce many detections. Furthermore, for better accuracy it may be beneficial to run other frames of different sizes across the image to counter variations of the object's size.

Figure 6.

This approach, however, introduces a new problem. After running each segment of the image through the bounding box regression model, there will be many bounding boxes that encapsulate no object (no object was in the segment) and possibly multiple bounding boxes for a single object. For the example above (figure 6), under the assumption that the color boxes represent the only segments of the image collected, there will be nine bounding boxes produced with only one that possibly captures the object. Therefore, in order to identify inaccurate bounding boxes a new learned metric, an objectiveness score is needed. Since the bounding box regression model was originally trained to have objects located near the center, it makes sense to have the objectiveness score represent the probability that an object is located roughly around the center. The objectiveness score can be learned by either appending a fifth output on the bounding box regression network, or creating a separate convolutional neural network that is also given the segment of the image and predicts this score alone. Now, having a bounding box and objectiveness score for each segment, the non-max suppression algorithm is a common approach to eliminate all but the desired bounding boxes. It does this by first eliminating all bounding boxes that have an objectiveness score below a set threshold. Next, sort all of the bounding boxes in descending order of their objectiveness score. Starting with the first (highest objectiveness score) go through all of the rest and eliminate those that have a high overlap (with the first), use IoU as a score to determine overlap. Repeat this process until there are no more bounding boxes to be eliminated. This approach gets rid of inaccurate bounding boxes as well as keeps only the best bounding box for each object in the image, resulting in multi-object detection.

Feature Extraction, Affinity Stage and Tracking -

After multi-object detection, the final task is to find a way to identify the same objects though frames. Say that we have detected the bounding boxes of two people in a frame, one wearing a red shirt and the other a green. In the next frame it is likely that both of those people are still present wearing the same colored shirts. Furthermore, unless the people are moving very fast, the camera is moving or the camera is very close to the people, it would be expected that the location of each person should be close to the location found in the previous frame. Therefore, the features (green shirt vs. red shirt as a simplified example) and the spatial locality of people between frames may be a good indication of identifying the same person. The influence of features compared to spatial locality on identification can be set as a hyperparameter depending on known factors(like a moving camera) of the problem.

Feature extraction can be as simple as a normalized pixel histogram, counting each occurrence of a pixel color for an object and dividing each channel by the number of pixels. Although there are better techniques, a histogram is sufficient enough for the purposes of this research. As for spatial locality, IoU is the most common approach. Using the bounding boxes from adjacent frames, the IoU and comparison of pixel histograms can be used to predict the probability that the objects are the same. This score is referred to as the similarity score.

Using all of the previous stages, tracking can now be implemented. Starting from the first frame, run the multi-object detection algorithm. For each detection label it with

an unique identification number and timestamp and compute the normalized histogram. Save the bounding box as well. Moving to the next frame, run the multi-object detection algorithm again. Compare the histograms and IoUs for each pair of detected people between the frames to compute a similarity score for each. For any pair of people with a similarity score above a threshold, label the current frame's detected person's ID to match the ID number of the detected person in the previous frame. Also save the timestamp and bounding box again. Alternatively, if a newly detected object has no affinity score higher than the threshold with any of the detections in the previous frame, then give it a unique identification number. Keep repeating this process for each frame in the video. Ultimately, there will be a list of detections, each with their location, timestamp and ID number. A simple algorithm can be used to connect the dots and compute each object's path taken.

Affinity Stage / localization stage

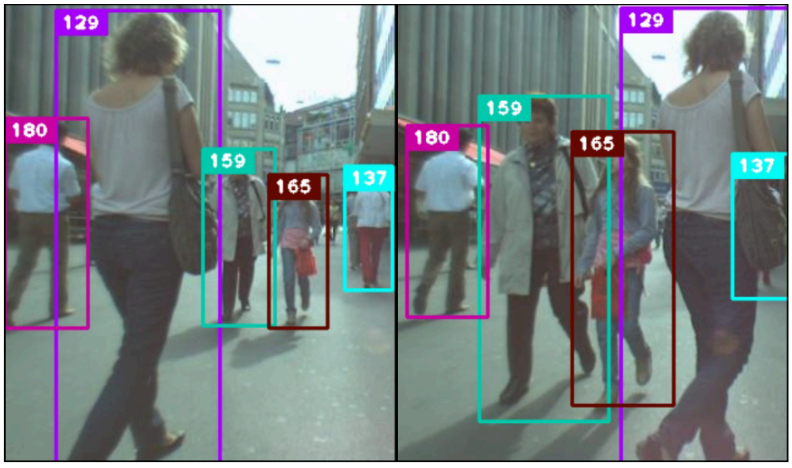


Figure 7.

Implementation and Results -

This project is implemented using Tensorflow and Keras in Python. (Tensorflow version - 2.13.) The dataset used is a collection of 847 images containing a person, downloaded from Kaggle.[3] The images are all different sizes and have a wide variety of backgrounds. Furthermore, a csv file is attached to the dataset containing the ground truth bounding box for each image, four integers representing the original image's pixel coordinates. The whole project is available on github to view or download. (https://github.com/TDanielsSkidmore/SkidmoreThesis)

In order to pre-process the data, the images are loaded into tensor arrays using the cv2 library and matched with the images' corresponding bounding box (from the csv file) and stored in a list. Each image is resized to (256,256,3) and all pixels are normalized to value between -1 and 1. It is preferable to have the image be relatively small and square since this model will be used with segments of photos (small and square) later in MOT. The bounding box coordinates are also normalized by dividing them by the original width or height of the image respectively. Lastly the data set is split into a training set (80%), validation set (10%) and test set (10%).

With the data now preprocessed, the convolutional neural network for bounding box regression will be described. Two implementations of networks are used here for comparison, the MobileNetV2 architecture and a self-designed architecture. My architecture consists of 11 convolutional layers, 6 pooling layers and 4 dense layers.

(including the output layer) Each convolutional layer uses a 3x3 mask size, padding to keep the output size the same as the input and the ReLU activation function to introduce non-linearity into the model. The architecture starts with 2 convolutional layers that produce a low dimensionality output. However, the size of each dimension of the output is large. The low dimensionality is to prevent the number of learned parameters from blowing up. Pooling layers are placed in between convolutional layers in order to decrease the size of each dimension from the convolutional layers. As the pooling layers do this, I increase the dimensionality of the next two convolutional layers to produce possible more learned different features. So, the pooling layers are used to allow us to increase the number of features learned without blowing up the number of learned parameters. This alternating pattern of convolutional layers followed by pooling layers are repeated several times. After the last convolutional layer, the data is flattened and passed through 3 dense layers gradually decreasing in dimensionality and using ReLU as the activation function. Dropout layers are also introduced throughout the dense layers to fight overfitting of the model. Finally, the output layer consists of 4 outputs with no activation function (any real number). Originally, I used the sigmoid activation function to keep the outputs in the range of 0 and 1 (which is the same range of the labels), but no activation function seems to work better. This may be because the model must learn what the outputs mean rather than restricting the model to only viable outputs. Note that it is now possible for the model to predict coordinates that do not actually exist in the photo. The total number of learned parameters is 3,262,590. For more information see myArchtecture.txt on github. The architecture is shown in figure 8.

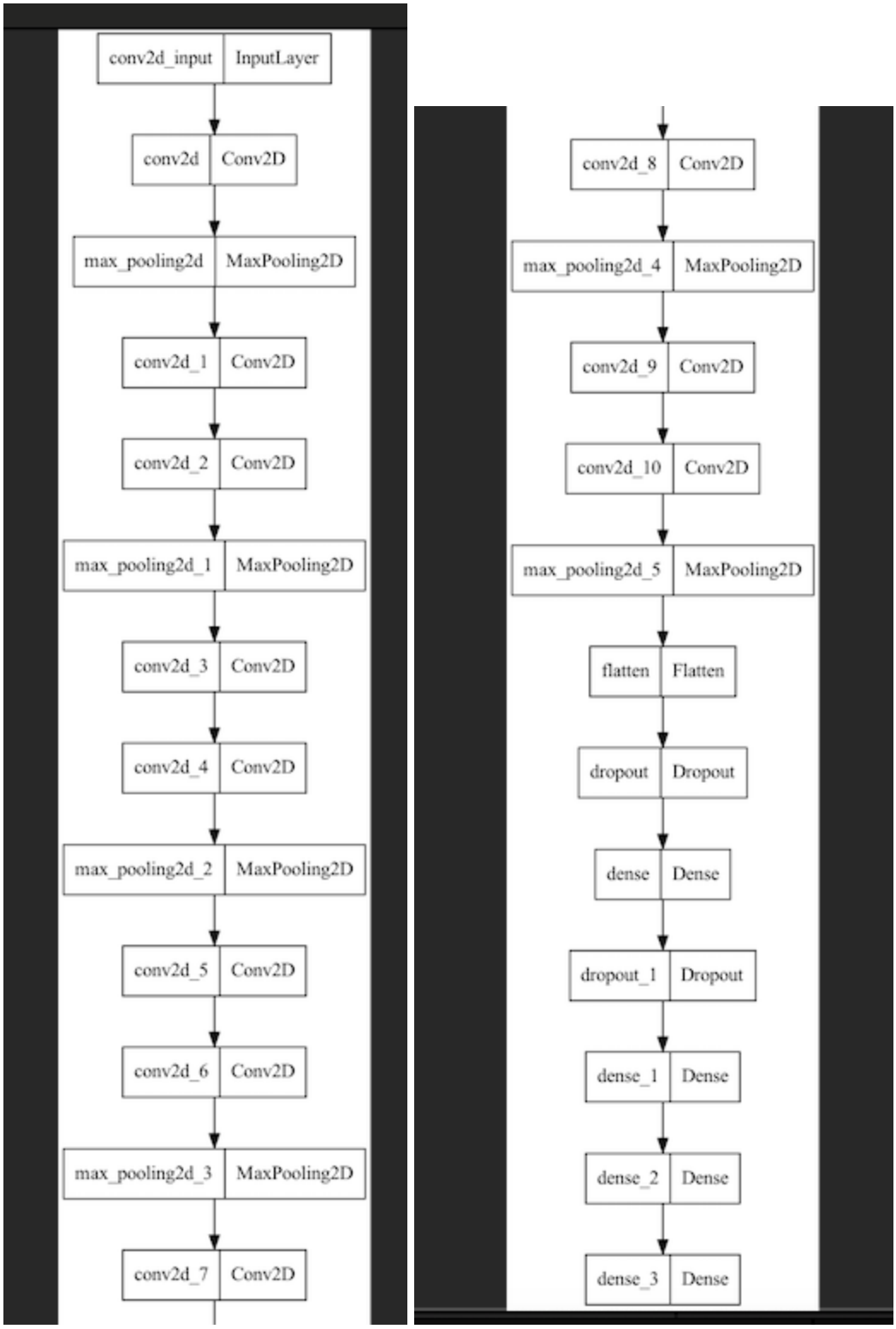

conv2d_input InputLayer
conv2d Conv2D
max_pooling2d MaxPooling2D
conv2d_1 Conv2D
conv2d_2 Conv2D
max_pooling2d_1 MaxPooling2D
conv2d_3 Conv2D
conv2d_4 Conv2D
max_pooling2d_2 MaxPooling2D
conv2d_5 Conv2D
conv2d_6 Conv2D
max_pooling2d_3 MaxPooling2D
conv2d_7 Conv2D
conv2d_8 Conv2D
max_pooling2d_4 MaxPooling2D
conv2d_9 Conv2D
conv2d_10 Conv2D
max_pooling2d_5 MaxPooling2D
flatten Flatten
dropout Dropout
dense Dense
dropout_1 Dropout
dense_1 Dense
dense_2 Dense
dense_3 Dense

Figure 8.

MobileNet is a noteworthy architecture published in 2018 and is designed to be a deep network but also lightweight and fast, which due to my computing and time constraints made it the perfect architecture to compare mine to. [2] [5] Two notable features in the MobileNet architecture are the use of depthwise separable convolution layers and skip connections. Depthwise separable convolution layers are designed to break up the data more in order to find spatial and cross-dimensional patterns based on certain dimensions of the data rather than being forced to find these patterns only through all the data. Skip connections is where an output of one layer earlier in the network is passed into a layer much further down the network (skipping the layers in between) which helps the network fight overfitting. This architecture is incredibly deep. There is a picture and summary of the architecture on github. In order to use MobilleNet for bounding box regression, I appended a convolutional and dense layer (with dropout) to the end of the architecture. The output is also four real numbers (no activation function). Despite MobileNet's depth, it only has 1,775,540 learned parameters. (which is why it is so lightweight and quick) I have attached a picture of a small section of the architecture in figure 9.

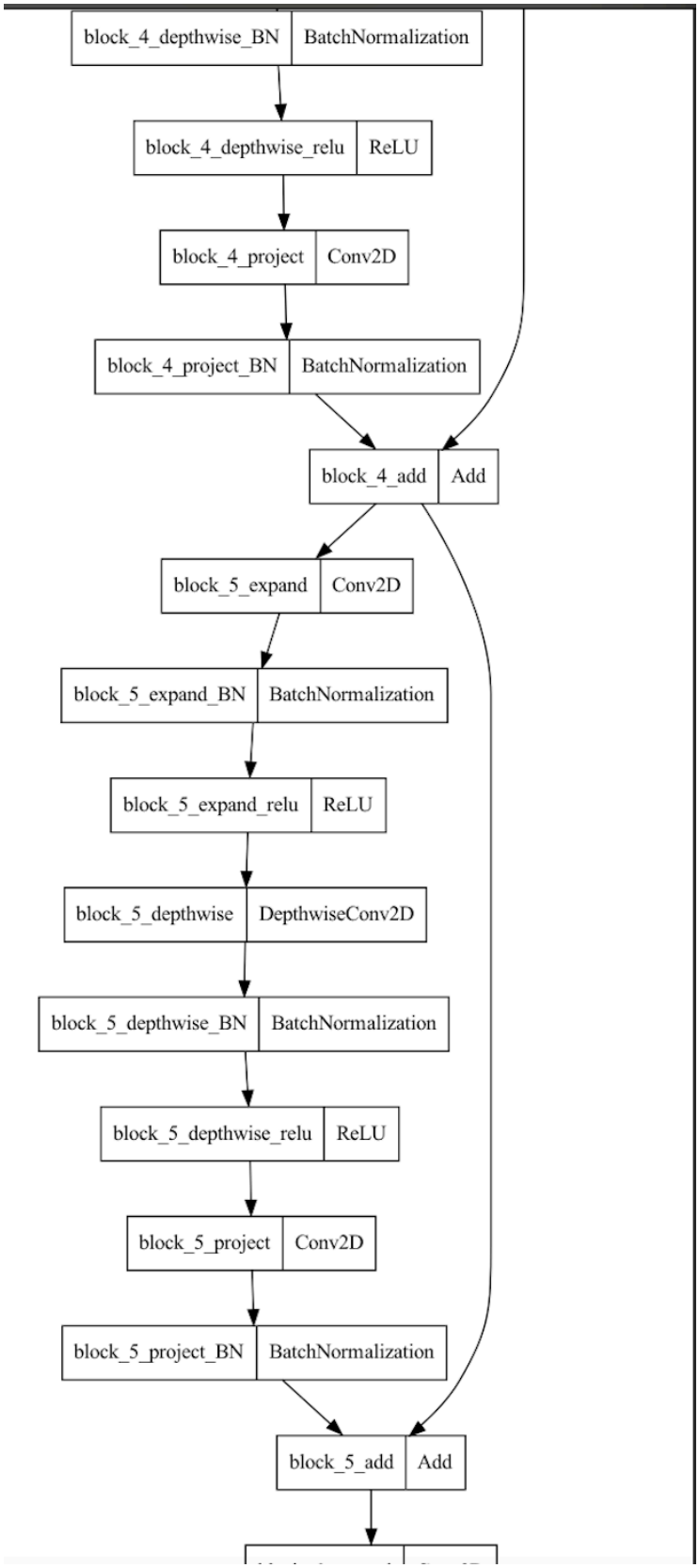

block_4_depthwise_BN
BatchNormalization
block_4_depthwise_relu
ReLU
block_4_project
Conv2D
block_4_project_BN
BatchNormalization
block_4_add
Add
block_5_expand
Conv2D
block_5_expand_BN
BatchNormalization
block_5_expand_relu
ReLU
block_5_depthwise
DepthwiseConv2D
block_5_depthwise_BN
BatchNormalization
block_5_depthwise_relu
ReLU
block_5_project
Conv2D
block_5_project_BN
BatchNormalization
block_5_add
Add

Figure 9.

Both MobileNet and my self-designed model are trained using the Adam optimizer (most commonly used one), a learning rate of 0.001 and mean squared error as the loss function. Training runs for a total of 500 epochs against the training data and it is checked against the validation set at the end of each epoch. Furthermore, using keras's model checkpoint function the weights are only saved for the trained model after an epoch if the validation loss is a new low. The results of the losses during training are shown below. The low validation loss, in both models, shows us that they were able to learn patterns from the data in order to predict bounding boxes. The main issue, especially for my model, is due to overfitting. The loss graph of my model shows a small, but significant, gap between the loss line and the val_loss line (figure 10). Although the gap is small the difference is relatively large. For example the values at the end of training are - loss = 0.0014 and val_loss = 0.0157. This shows us that my model had a tough time generalizing the problem. It is tough to tell since MobileNet's loss graph has a larger range on the y-axis, but it did a better job at fighting overfitting (figure 11). Upon analyzing the values at each epoch I found that at some points it reached the values of - loss = 0.0059 and val_loss = 0.0081. Which shows a significant improvement of generalization compared to my model. However, it seems that MobileNet could not generalize past this point. The fluctuation of the val_loss line toward the end of the graph shows that it began to seriously overfit the data as the loss decreased more.

My Model's loss graph

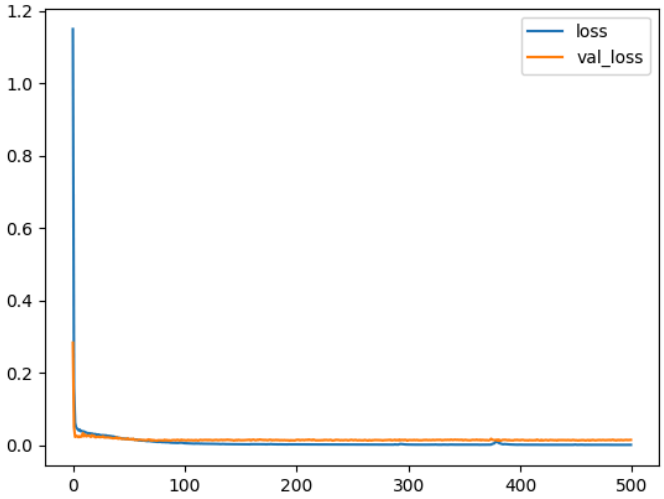


Figure 10.

MobileNet's loss graph

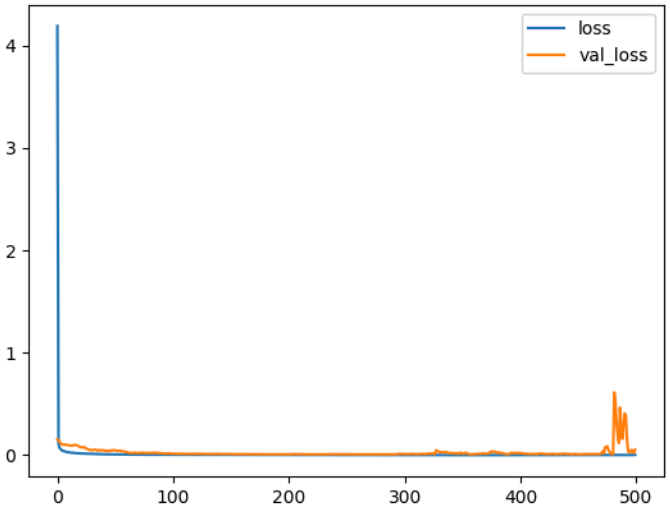


Figure 11.

Looking at some of the results of the model when given the test set, both models generally are able to set a bounding box that captures most of the person in an image (figure 12/13). Some of the test examples are shown below, including the only completely missed bounding box (image 4 in figure 12/13). The blue boxes are the ground truth bounding boxes while the green are the predicted.

My Model

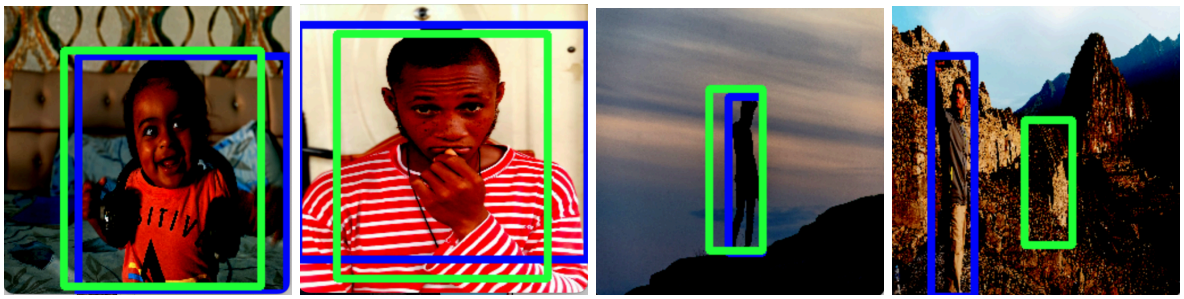


Figure 12.

MobileNet

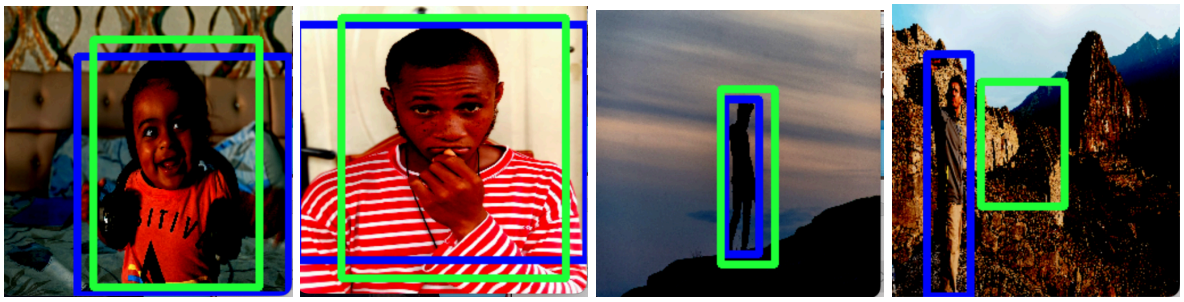


Figure 13.

Other than the visual representation of the outputs of the models, we have no metric that indicates how well the models accomplish their task (bounding box regression). For example, the low mean squared loss obtained from training tells us that

the model did learn to fit the data, but that metric (0.0157 and 0.0081) gives us no reliable expectation of the model. So, intersection over union (IoU) is used to evaluate the model (figure 14). Using the test set from the original data, each image is passed through the model and computes the predicted bounding box. The predicted and ground truth bounding boxes are then used to calculate the IoU. The IoU score is averaged out over all of the testing data.

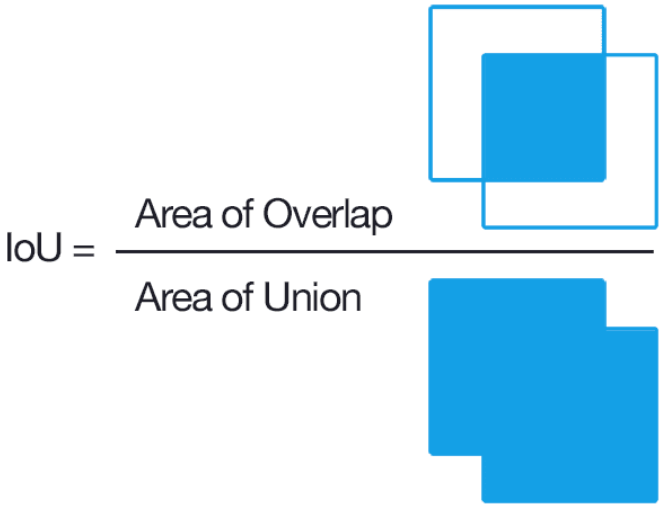


Figure 14.

In order to compute the IoU, the predicted bounding box values must be set to the range (0,1) since it should not be possible to have a coordinate outside the image and the model is not forced to output values in this range (could be any real number). The simplest way to do this is to set any value over 1 to 1 and any value under 0 to 0. Upon using code I found in a resource to compute the IoU, I began to see IoU values over 1 which should not be possible since the overlap cannot be more than the union. So, I created my own way of calculating the IoU -

Given the ground truth bounding box (x1,y1,x2,y2) and predicted bounding box ($\hat{X}1$,$\hat{y}1$,$\hat{X}2$,$\hat{y}2$) find the rightmost left vertical edge (T1), downmost upper horizontal edge (T2), leftmost right vertical edge (T3) and upmost down horizontal edge (T4). These four values will give you the coordinates of the intersection as long as there is an intersection. Then choose one bounding box as reference (I use the ground truth), to see if the coordinates (T1,T2,T3,T4) fall outside (left, right, up or down) of that box. The process is as follows:

T1 = max(X1,$\hat{X}1$)

T2 = max(Y1,$\hat{y}1$)

T3 = min(X2,$\hat{X}2$)

T4 = min(Y2,$\hat{y}2$)

Check to see if any of the conditions T1>X2 or T2>Y2 or T3<X1 or T4<Y1 are true. If so that means that there is no intersection and the actual intersection area should be set to 0. Otherwise, set the intersection to be the area of the coordinates.

Intersection = (T3-T1) * (T4 - T2).

To calculate the union, subtract the intersection from the sum of the two bounding box areas.

Union = (X2-X1)(Y2-Y1) + ($\hat{X}2$ - $\hat{X}1$)($\hat{y}2$ - $\hat{y}1$) - intersection

Then, IoU = Intersection / Union

Calculating IoU this way eliminated all of the erroneous results that were present before. Using this calculation to evaluate the model's predictions on the test data, these were the average IoUs of the models.

MyModel = The average intersection over union is 0.591 with the standard deviation of 0.00345

MobileNet = The average intersection over union is 0.699 with the standard deviation of 0.00532

This shows that both models performed adequately, MobileNet performed significantly better, in that each bounding box predicted on average captures at least most (over half) of the person and that the size of the bounding box is relatively correct. Moving on from a functional bounding box regression model, the next step is to generalize the problem to capture bounding boxes of several people in one photo. As discussed in the Multi-Object Detection chapter, we need to train a new model to predict the objectiveness score. Although it is typical to train an objectiveness score model to predict a value indicating how well the person was located in the center of the image, I have generalized the problem to predict a value of how well the person was located anywhere in the image. This change is necessary since our bounding box regression was also not trained with images with a person located roughly in the center. In other

words, the objectiveness score (or the value of how well the input image is fitted such that the bounding box regression model can produce values that capture the best possible bounding box for a person) should be trained on data similar to what the bounding box regression model is expecting. However, the current training data used for bounding box regression is not directly useful in training the objectiveness score model. But, we can preprocess the data in order to change it to what we need. The data currently has one whole person (or most of a person) per image and 4 coordinates representing the ground truth bounding box. We need a data set that contains different amounts of a person in each image (from no person to a full person) and labels representing the amount of the person in that image. We can create this data set from segmenting each image in the current data set into several images. We can get the label by computing the intersection of the ground truth bounding box and the coordinates of the segment. The intersection calculation from above is applicable here. Using simple “for” loops, I slide what can be visualized as a filter over the image (across and downward) where each iteration of the filter produces a new segment. The size of each segment is (1/2 of the image height, 1/2 of the image width) and then is shifted by 1/6th of the image each time. Here is a representation of the new data (figure 15, red box is the new segment) with the labels, respectively, 0.104, 0.339, 0.339, 0.199, 0.652.

Figure 15.

After preprocessing the data in this way, the previous networks can be used with a small change in the output– the output layer needs to be changed to produce only one number. Furthermore, I felt that this problem is a simpler one than the previous bounding box problem. (Just detection no localization) So, I originally decreased the learnable parameters of my model by decreasing the dimensions of the output at each layer. Upon training the model, It was able to reach a low loss but when used against testing data, it produced the same prediction regardless of the input. This indicated to me that the model was not learning at all. I began by analyzing the data I created and soon realized the possible biases in the data. I found that the distribution of possible labels (and the data) was skewed towards a specific range of values depending on the size of the segments. (the larger the segment the larger the labels typically are) Analyzing it further, I found for my current segment size that nearly 70% of the data fell within close to 40% of the full range of labels. (0.15-0.54) Furthermore, the segments taken from an image were saved sequentially which may have grouped similar data. All of this may have prevented the model from learning and instead allowed it to find a

number which worked well with the skewed data. I was able to confirm this by increasing the segment size and noticing an increase in the predicted value. After further processing of the data to bring it closer to a uniform distribution (based on the labels) and shuffling the data around, I began to see some variation in the predictions when the model was trained over a very small number of epochs but the variation was lost under further training. I even tried to train my initial architecture (with the full amount of parameters) to no avail. However, when I trained MobileNet under the same data I saw great results. Setting the model's loss to mean squared loss, learning rate to 0.001, optimizer to Adam and training it over 500 epochs, the model was able to predict the testing data to within 0.029, on average, from the true label. The loss graph is presented in figure 16.

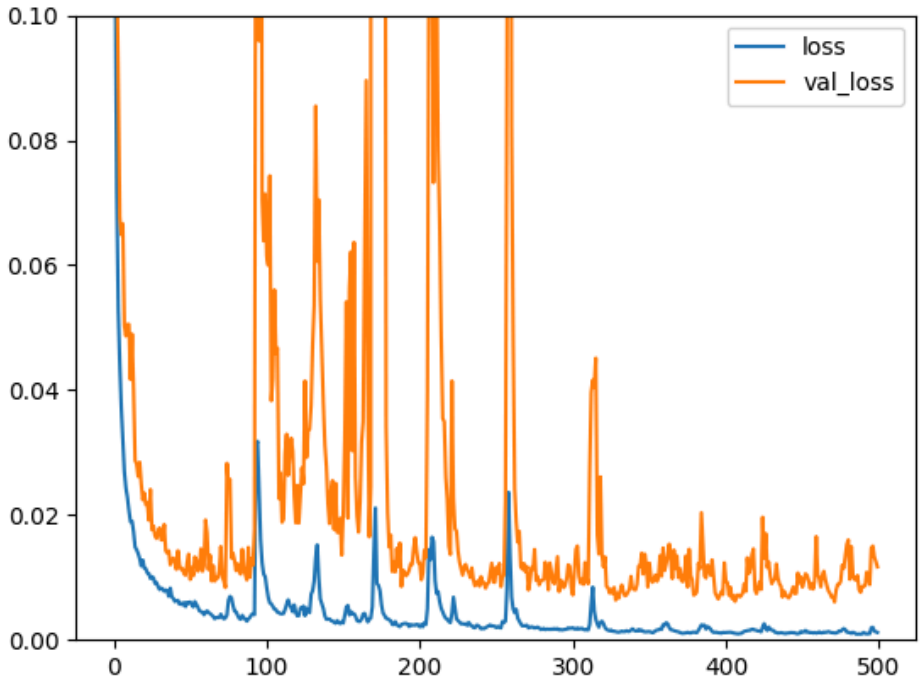


Figure 16.

It would seem that my model is just incapable of learning the patterns needed to perform this task. It may be possible that my network is just not deep enough to uncover the low level patterns needed. Regardless, MobileNet gives a great model to use for the objectiveness score.

With trained models for bounding box regression and objectiveness scores, multi-object detection is now possible. A new dataset of images with multiple people per image is required. However, since there is no more training needed to accomplish multi-object detection, I made a small data set for testing. So, I used pictures of family and friends with various numbers of people in each image. For example, I used a picture of my two nieces at the beach. The first step is to segment the image again (sliding filters) in order to create images to pass to each trained model. Since we do not know the size of each person (how close they are to the camera) creating segments of different sizes (smaller and larger filters) will help us generalize this better. When sliding the filter across and down the image we do not want to skip over people so the skip value needs to be small in comparison to the width and height of the image. Additionally, since both models need a square image (256 by 256), I decided to select segments that were also square. If a segment is not square when it is reshaped the people may be distorted in some manner and then negatively affect the predictions of the model. I created filters of several sizes:

Filter1 = min(# of rows / 1.2 , # of column / 1.2)

Filter2 = min(# of rows / 1.5 , # of column / 1.5)

Filter1 = min(# of rows / 2 , # of column / 2)

Filter1 = min(# of rows / 3 , # of column / 3)

Horizontal skip value = # of columns / 16

Vertical skip value = # of rows /16

Each segment is then pre-processed into image sizes of (256,256,3) and all color channels are normalized between (-1,1) which creates images of the same shape and value range the model requires. Furthermore, the end goal is to create bounding boxes for each person in the original image, so the location of each segment in relation to the whole image cannot be lost. For each segment the original segment size and the position of the segment's origin (top left pixel coordinates) in the original image is saved. Now, each segment is passed into the models to predict a bounding box and objectiveness score for each. All that is left is to determine which bounding boxes should be deleted. In order to do this, I used the non-max suppression algorithm which works as follows. Set a threshold value for the objectiveness score and get rid of any bounding box with the objectiveness score lower than that threshold. (get rid of any bounding box taken from a segment of which is expected to have little to none of a person in it) Next, sort each bounding box in descending order based on their objectiveness scores. Starting at the first bounding box in the sorted list (max objectiveness score) and using another threshold, get rid of any bounding box that has a IoU higher than the threshold with the bounding box you are currently on (best

objectiveness scores). Iterate over the whole sorted list until the end is reached. This is done because we wish to just have one bounding box per person. After the first step, the hope is that the only bounding boxes left are ones which are at or around each person (as the objectiveness score for that segment indicates a person). Then starting at the bounding box derived from the segment with the best expected amount of people in it, get rid of any other bounding box around that person. What is left over should be the best bounding box for each person. Lastly, after the non-max suppression algorithm runs, each bounding box needs to be mapped to the original image. In order to do this, I multiplied each value in the predicted bounding box by the original segment's length (square dimensions) and then added the horizontal and vertical offset of the segment's origin in relation the original image to the bounding boxes values respectively.

I quickly realized a problem with using IoU as a filtering metric in the non-max suppression algorithm. If one smaller bounding box is within a larger bounding box, it is possible that the IoU score may be small. This may happen because although the intersection is the largest possible, the larger bounding box creates a large union driving down the value (regardless of the intersection). For example, examine a part of an image I ran multi-object detection, see figure 17.

Figure 17.

The IoU for these two bounding boxes was 0.567 and less than my threshold of 0.6. So, this bounding box appeared in my results although it should be eliminated since we do not want two bounding boxes potentially capturing the same object. In response to this, I created my own metric I call normalized overlap. It is simply the intersection of the two bounding boxes divided by the area of the smaller bounding box.

Normalized Overlap = intersection / min(area of bounding box 1, area of bounding box 2)

After implementing this new metric in the non-max suppression algorithm I saw such cases eliminated in the final results. Here are some images I ran multi-object detection on with a 0.6 objectiveness threshold and 0.4 normalized overlap threshold (figure 18).

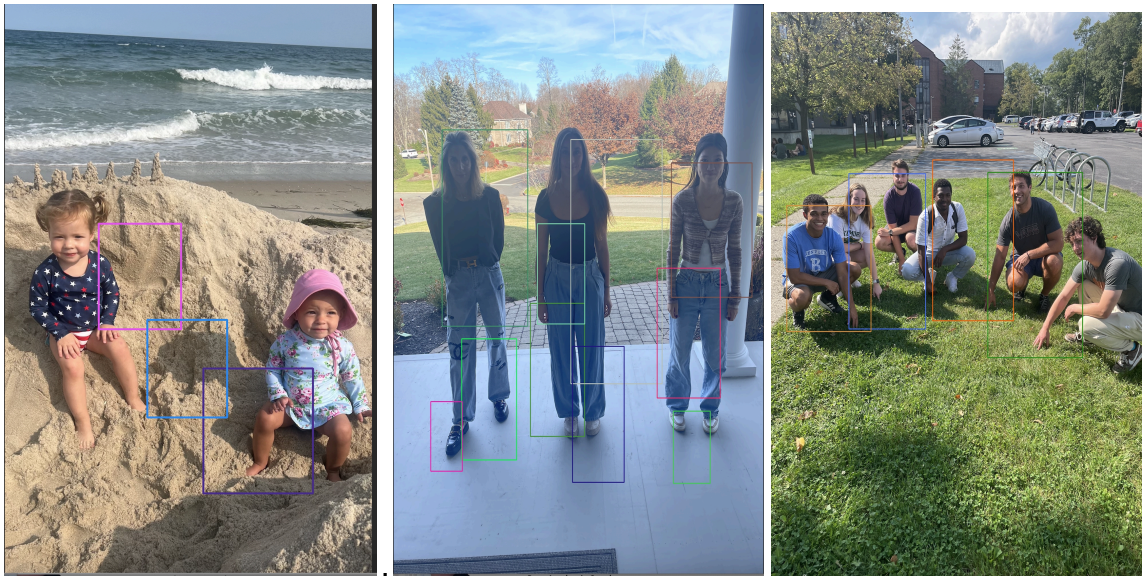


Figure 18.

Although some of these results are promising, overall the results were rather disappointing. In the first image, we see almost no correlation between the bounding boxes and the children. In the second and third image, the bounding boxes are in the general area of where they should be but do not capture people very well. I contribute these disappointing results to two problems in the current multi-object detection. First, it is clear that the bounding box regression model doesn't predict bounding boxes that capture a person well enough and must be improved. Secondly, there is a disconnect between the objectiveness score and the bounding box. The objectiveness score judges the capability a bounding box regression model has to locate a full person in a segment, but that doesn't mean that the model actually does a good job at locating them. As a result, it becomes very possible that a bad bounding box gets treated as a good one (high objectiveness score in a segment but the model fails to locate it). This bad bounding box with a high objectiveness score may be even used to get rid of possibly

good bounding boxes. For example, suppose a segment containing an entire person is given to the objectiveness score model and the model predicts a very high score. But, when the segment is given to the bounding box regression model, the model fails to capture the whole person and only captures a section of them. Since this segment was given a very high score, it is likely that it will be used against (to get rid of) other segments with possibly perfect bounding boxes.

I propose an alternative approach to the objectiveness score. Rather than having a score for the segment, have a score for the bounding box itself. If an objectiveness model can accurately learn this, then the disconnect from the objectiveness score and bounding box would be eliminated and in theory the results improved. The only issue in training such a model is the fact that the bounding boxes are not a consistent size or shape. Reshaping the bounding boxes may again distort the image and person making it very hard to learn. However, passing in the original segment with every pixel outside of the bounding box set to black (0,0,0) should have the same effect without the distortion of the image. Using the original data set, I segmented the images just as before and passed them into the bounding box regression model to get a predicted bounding box for each segment. I then blacked out any pixel in the segment not located within the predicted bounding box. I used the IoU metric for the ground truth and predicted bounding boxes as the label for the new data. Obtaining training data like the examples in figure 19 with the labels respectively: 0.109, 0.251, 0.227, 0.0, 0.651.

Figure 19.

I trained this new model on an identical architecture (MobileNet) as used for the original objectiveness score with the same hyperparameters but with this new data and it was able to achieve a low loss score (figure 20). The average distance of the model's predictions from the true values when given the training set is 0.172, which is not as good as hoped, but is close enough to be able to evaluate each bounding box.

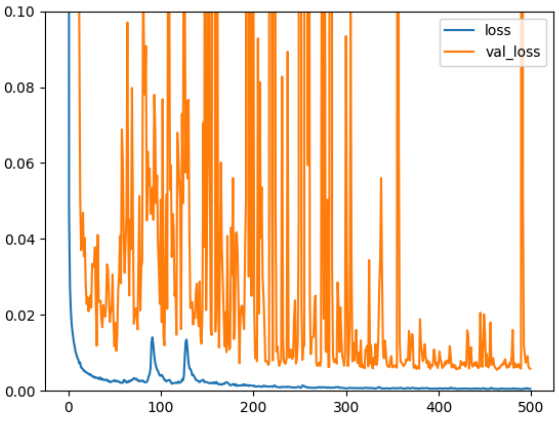
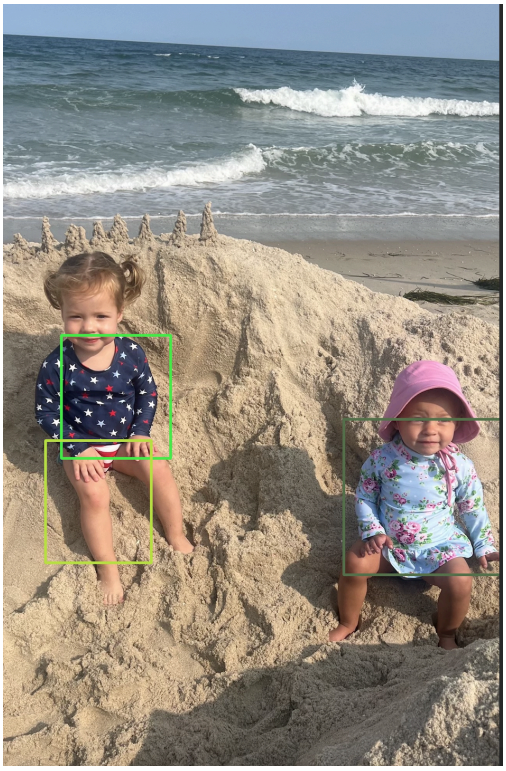
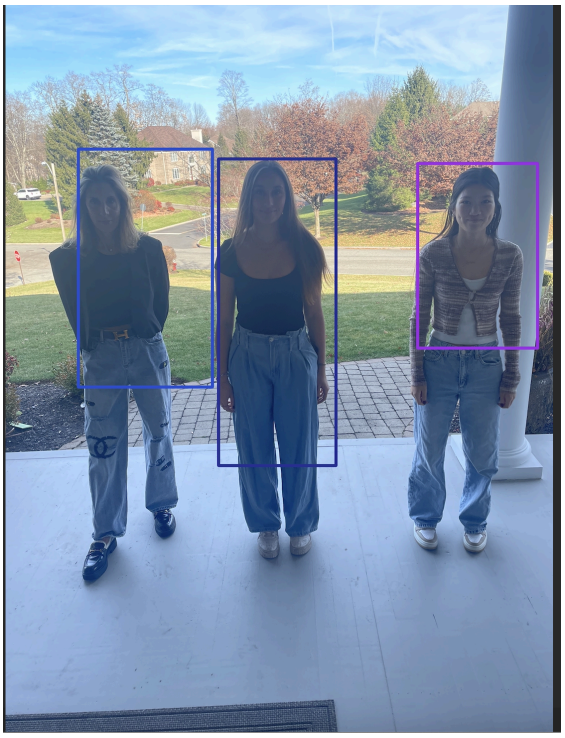


Figure 20.

Using this new model for multi-object detection, I was able to obtain much better results. Nearly all bounding boxes that do not capture some section of a person are

eliminated and it seems that the best bounding boxes are the ones saved. One thing to note is that when the objectiveness score threshold is set high (0.7 and above) no bounding boxes are generally saved. (No bounding boxes reach an objectiveness score this high) This is still caused by the inaccuracy of the bounding box regression model. The best results from objectiveness thresholds of a range from (0.3-0.5) and normalized overlap thresholds of a range from (0.2-0.4) are presented below in figure 21.





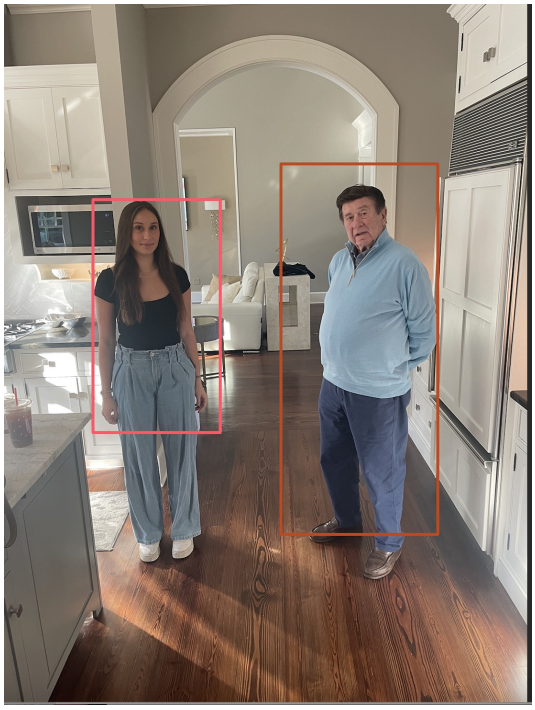


Figure 21.

As some of the results indicate, many of the bounding boxes do not grab a whole person, but only a section of them. This may be due to the bounding box regression model but upon further inspection, it looks unlikely. Looking at the bounding boxes in question, many of them are square shaped or close to it. It would make sense that it is the square segments causing this rather than the bounding box regression model. In order to test this theory, I changed the segments to match the shape of the original image (no longer square) and then resized them back into the expected format (square) , understanding the possibility of more error due to a distortion of the segment (and each person). Upon analysis of the results, I was able to confirm that the square segments were having a negative effect of bounding whole people. Furthermore, it seems that any distortion of the image had no effect on the accuracy of the bounding box model. The results are presented below in figure 22.

Figure 22.

Future work-

In the future I hope to first improve the bounding box regression model. In order to continue to build a successful multi-object tracking software a better bounding box regression model is needed. Right now the inaccuracy in the bounding box would make the similarity scores in the next part of multi-object tracking also very inaccurate. I plan on trying different Convolutional Neural Network architectures, hyperparameters and preprocessing to build the best model possible. Furthermore, I hope to find solutions that will segment the image better. Although there was improvement making the segment sizes adaptive to the image's shape, there may be other ways to segment the image which will have even more success. I plan on trying many different shapes and sizes for the segments in order to find the best ones. Once these routes have been explored, I will complete multi-object tracking software by implementing the feature extraction and the affinity score stages.

Bibliography-

1. Athar, A., Luiten, J., Voigtlaender, P., Khurana, T., Dave, A., Leibe, B., & Ramanan, D. (2023). BURST: a benchmark for unifying object recognition, segmentation and tracking in video. *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. https://doi.org/10.1109/wacv56688.2023.00172
2. Géron, A. (2022). *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems*.
3. *Human detection*. (2023, January 16). Kaggle. https://www.kaggle.com/datasets/nikhilroxtomar/human-detection?rvi=1
4. Rosebrock, A. (2023, June 9). *Object detection: Bounding box regression with Keras, TensorFlow, and Deep Learning - PyImageSearch*. PyImageSearch. https://pyimagesearch.com/2020/10/05/object-detection-bounding-box-regression-with-keras-tensorflow-and-deep-learning/
5. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *arXiv:1801.04381*. https://doi.org/10.1109/cvpr.2018.00474
6. Sun, P., Cao, J., Jiang, Y., Yuan, Z., Bai, S., Kitani, K. M., & Luo, P. (2022). DanceTrack: Multi-Object tracking in uniform appearance and diverse motion. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr52688.2022.02032
7. *TensorFlow text processing tutorials*. (n.d.). TensorFlow. https://www.tensorflow.org/text/tutorials

8. Wojke, N., Bewley, A., & Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. *University of Koblenz-Landau*. https://doi.org/10.1109/icip.2017.8296962
9. Wood, L. (2023, April 8). *Object Detection with KerasCV*. Keras.io. Retrieved November 27, 2023, from https://keras.io/guides/keras_cv/object_detection_keras_cv/
10. Xie, F., Wang, Q., Wang, G., Cao, Y., Yang, W., & Zeng, W. (2022). Correlation-Aware deep tracking. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr52688.2022.00855