COMPUTER VISION LAB: ASSIGNMENT 3

A PREPRINT

Merve Gül Kantarcı ID: 21527104
b21527104@cs.hacettepe.edu.tr

May 31, 2019

1 Introduction

This report explains a study on single object tracking. The program written is expected to track object through images which are frames of a video. Train dataset includes 12857 frames from 263 videos. Test and validation sets include 25 videos for each.

Two network is used mainly, one of them extracts features, other one predicts bounding box. Parameter tuning and data processing are reported.

Extra Libraries Pytorch, Pillow, tqdm, imageio, numpy, matplotlib

2 Implementation Details

Figure 1: Bounding Box after cropping

VGG16 network's last feature layer is changed as Average Pool 2D layer for extracting features. For the train and validation data, image paths are extracted and obtained pairs. Each frame is cut to 2x enlarged bounding box from center. Then it is resized and normalized to $[-1, 1]$ range. According to find the relative true coordinates for processed image, the formula is applied: $new\_coord = (old\_coord - corner\_coord) * (new\_size/old\_size)$ Modified VGG16 network extracts 1x512 vector from each processed image, and concatenating the two of them gives a 1x1024 sized vector. After extracting each feature, the relative coordinate labels, and features are saved as a tensor file in the same directory for decreasing time elapsed in the next run.

Another network for predicting bounding boxes is created with layers: FC (1024x1024), ReLU, Dropout(0.5), FC (1024x1024), ReLU, Dropout(0.5) and FC (1024x4). Network is initialized with random weights and trained from scratch. Data is shuffled in each epoch to make sure every neuron learns the all cases. Validation data is used to measure how well the network learns. By taking the weights that gives lowest validation loss, it is avoided to get the overfitting network. The best model is saved in the directory, in order to use it later without training again. However, this process does not take too much time. Training and validation loss is plotted for review. Test image frame pairs are extracted in a similar way to train and validation pairs, however it is not shuffled. For every test pair, starting from true coordinates, new coordinates of the bounding box is predicted. The prediction is done by the trained custom network and feature extractor is modified VGG16 as usual. For the first pair ground truth is used for getting track/look regions, while the rest uses predicted coordinates. Loss for every video is given in output. In order to draw the predicted bounding boxes, coordinates that are obtained from the network are considered as relative coordinates and their real position in original image frames are found with the formula $real\_coord = (relative\_coord * (new\_size/old\_size)) + corner\_coord$ GIF files showing predicted and ground truth bounding boxes on frames are exported to a folder named "results/" in the same directory for visualizing the track.

3 Problems Encountered

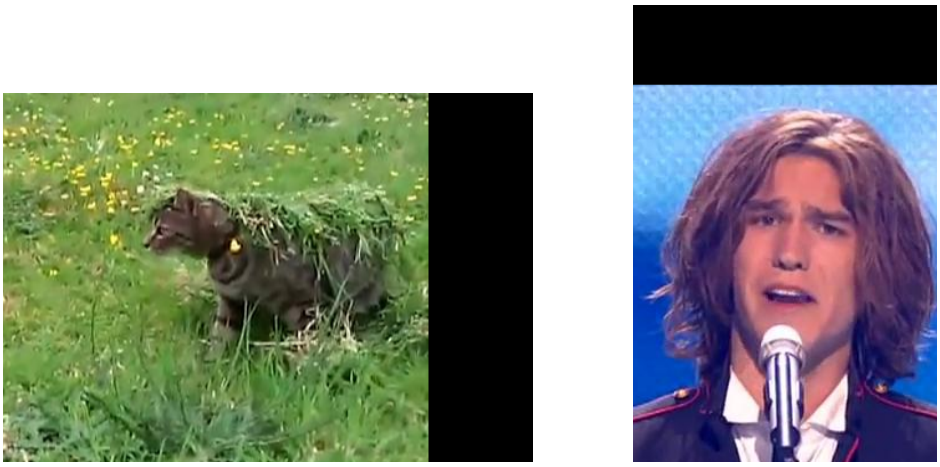
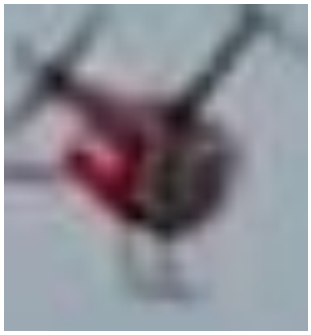


Figure 2: Out of dimensions for cropping

Out of Dimension for cropping When the wanted crop coordinates are out of dimensions, PIL library crops any way and pads the space with black region. This effects the feature extraction and testing part. In order to avoid this when the dimensions are exceeded, it is basically cropped until the boundaries (e.g. until 0,0 for right-top corner).

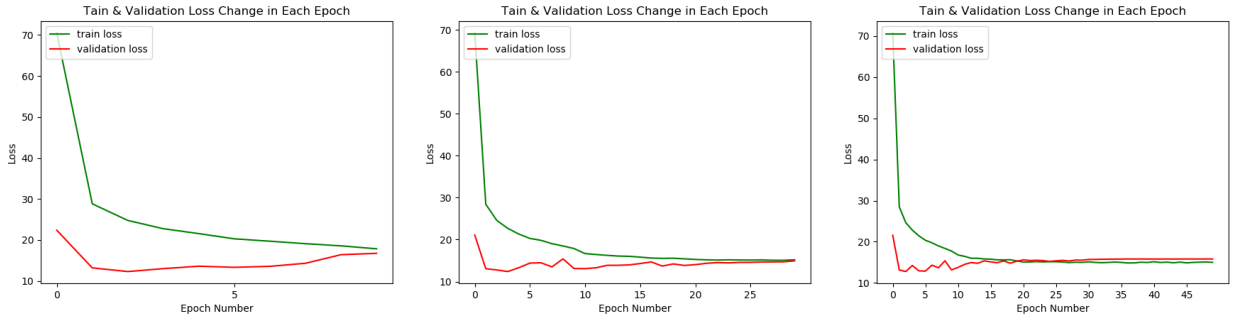
Shrinking Bounding Box For some of the cases, when the track of object is lost, network focuses on some small motion and bounding box gets smaller with it. So small that the cropped image becomes zero in one of the dimensions. In order to make sure that network keeps looking for the track, when 2x enlarged bounding box size gets smaller than 10 pixels in one dimension, it is cropped again by 4x enlarged bounding box. This problem was very rare that I observed 4 or 5 times in whole test set but this is the solution I finally came up with.



Resizing Firstly, resize amount was 220. Then after reviewing cropped and resized images, I realized this is a big scale for most of the cases. Because for some images, resized version was very blurry. This lead to experimenting on different scales, the results can be found in experiments section.

4 Experiments and Evaluation

4.1 Epoch Size



(a) lr = 0,0001 (b) lr = 0,0001 (c) lr = 0,001

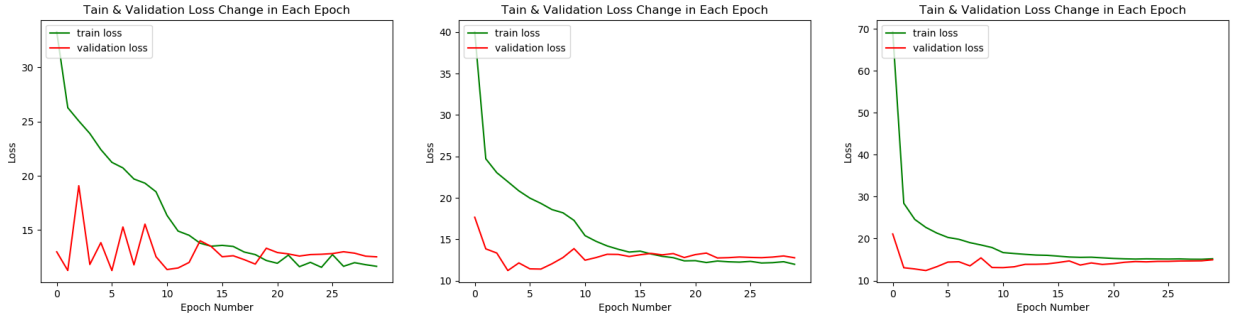
Figure 3: Different epoch size

Different epoch sizes are tried to see if there will be any significant changes on increasing epoch. Apparently 10 is very low and 50 epochs are unnecessarily high. It is decided to go on with 30 epochs.

4.2 Batch Size

Batch Size	Time Elapsed
16	7 m 52 s
64	2 m 4 s
256	0 m 37 s

Table 1: Batch size effect on time



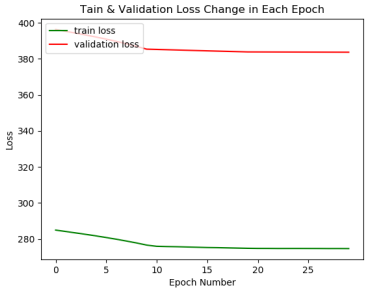
(a) Batch 16 (b) Batch 64 (c) Batch 256

Figure 4: Effect of batch size on loss

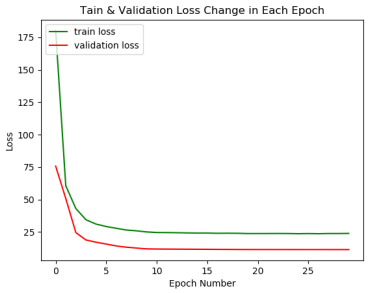
The accuracy are so similar that I could not observe a distinct effect of batch size on accuracy. However, reducing batch size increases the iteration count hence the time increases drastically.
However, when the figure is inspected, it can be observed that small batch leans more on memorizing the train data

instead of generalizing. The validation line is not steady. I decided on the batch size as 256 because it looks more steady.

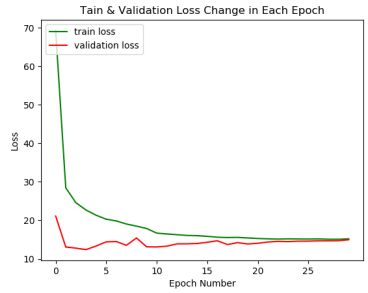
4.3 Learning Rate



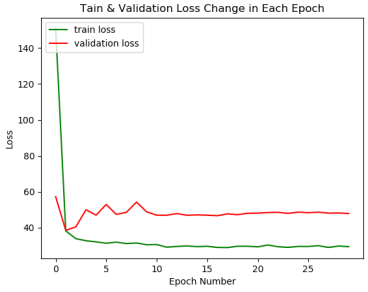
(a) Lr = 0.000001



(b) Lr = 0.0001



(c) Lr = 0.001

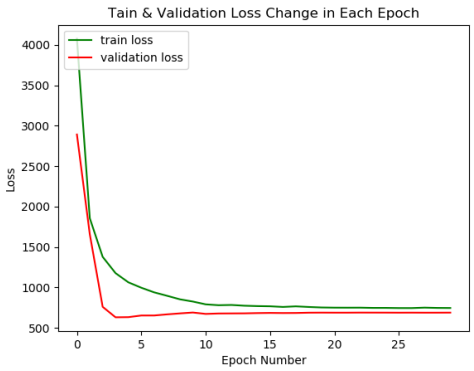


(d) Lr = 0.01

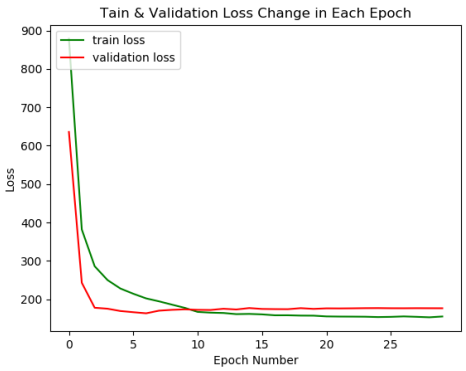
Figure 5: Effect of Learning Rate on Loss

Learning rate states how much a big step is going to be taken by the gradient descent algorithm. As it seen from the graphs, a learning rate that as big as 0,01 misses the minimum. The other two converges very slow or does not converge. It is decided $lr = 0,001$ would be a more steady choice.

4.4 Resizing

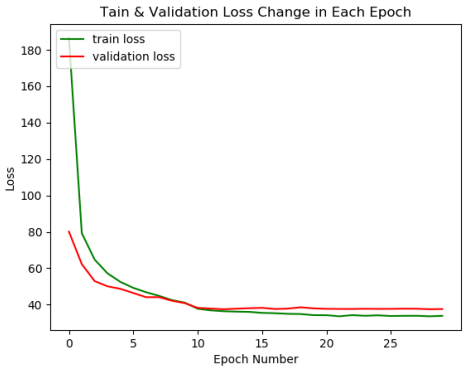


(a) Resize 120

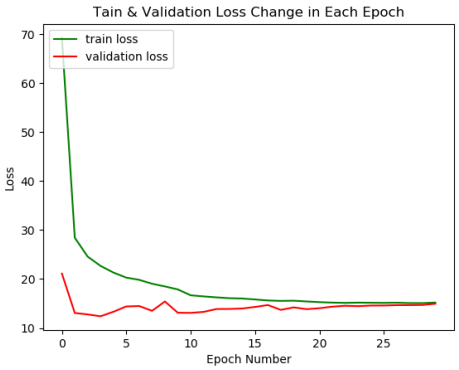


(b) Resize 60

Figure 6: Effect of Resizing on Loss (Bigger Scales)



(a) Resize 30



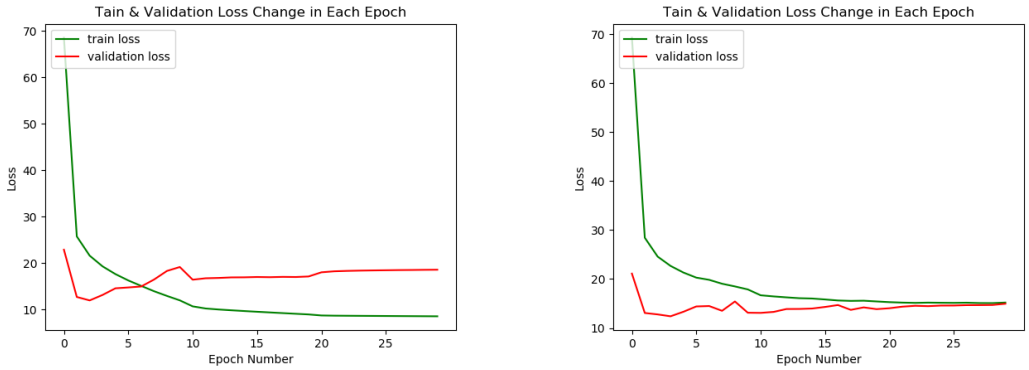
(b) Resize 20

Figure 7: Effect of Resizing on Loss (Smaller Scales)

This was the most challenging part in the experiments. Resizing decreased the loss significantly. And as it can be observed from 120-resized results, training loss was higher than validation loss. I concluded from that; data was not enough representative with 120-resized. Also effects of resizing to 220 makes some of the images very blurry since it makes the size bigger. However, decreasing loss could have mean that it is harder to make a wrong guess in a small scaled image for the network because area is really small. It is a bit misleading in this context. But visualizing the bounding boxes as GIFs made sure that small scale is more efficient.

4.5 Drop-out Layer

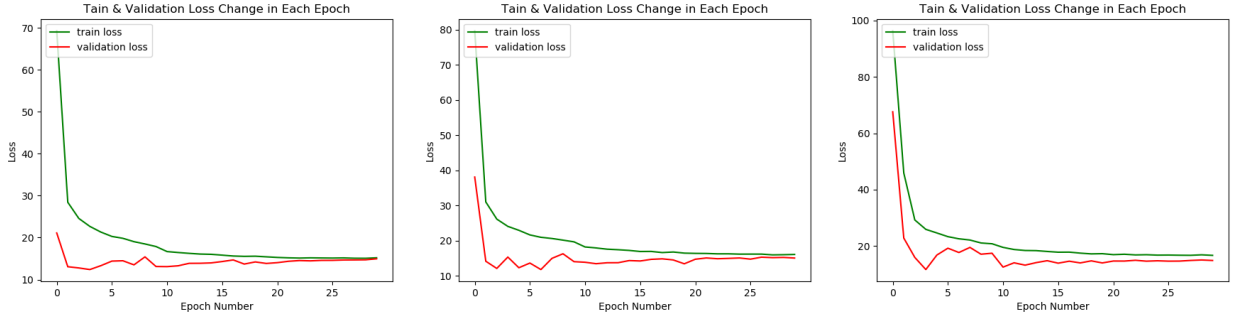
Drop-out helps network to generalize the data better. This can be seen from the plots above. Between epochs 5 and 10, firstly it seems to be learning similarly well, however peak in validation shows that there is a possible memorizing on train data. For the same input, without dropout train loss decreases to 8 while it decreases to 15 with dropout layers. This shows a possible memorization.



(a) Without Dropout (b) With Dropout

Figure 8: Effect of Dropout on Loss

4.6 Extra FC Layers

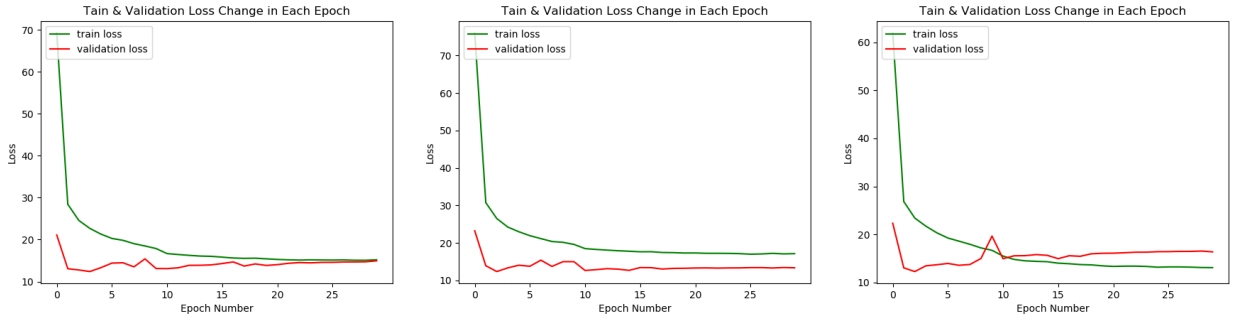


(a) Regular (b) Extra ReLU-Droput-FC (c) Extra 2 ReLU-Droput-FC

Figure 9: Effect of extra layers on loss

It did not seem to be a good idea adding 2 more ReLU-Dropout-FC layers by observing from loss graphs that it is not very steady. However, adding one more layer did not make a distinct change.

4.7 Changing Size of Hidden Layers

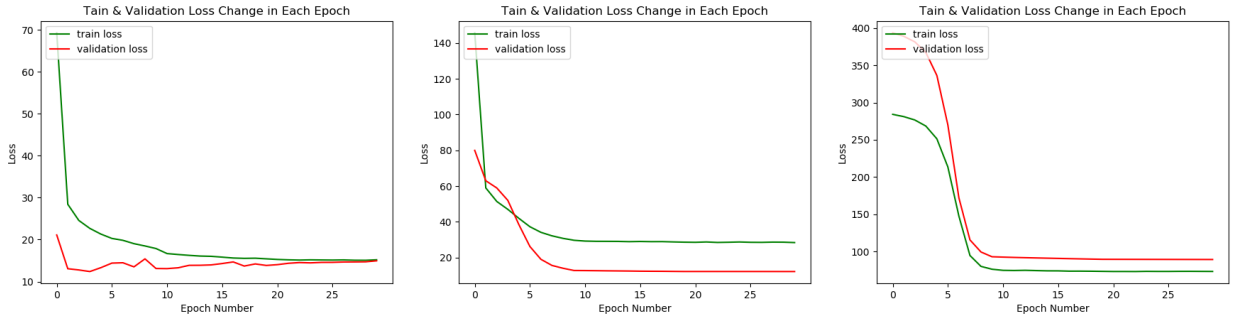


(a) Regular (b) First Layers 1024x1024 1024x512 (c) First Layers 1024x2048 2048x1024

Figure 10: Effect of hidden layer dimensions on loss

For the first and third case final accuracy results are similar. I found the third case also logical. It seems train data is learnt better than validation at third case, which is expected from a neural network. Second case has a disturbingly high difference between train and validation loss.

4.7.1 Different Optimizers



(a) ADAM (b) SGD (c) AdaDelta

Figure 11: Effect of different optimizers on loss

The best results are obtained with Adam optimizer. Other optimizers were significantly bad with the current adjustments (3 FC Layers with ReLU and Dropout) and hyper-parameters (Learning rate = 0.001, Batch Size = 256). AdaDelta seems like learning train data better compared to validation data, it might be promising with parameter tuning.

5 Conclusion

Video No	Loss	Video No	Loss	Video No	Loss
2	15562	**71**	1103	**119**	388342
16	2753	**73**	111976	**139**	30406
18	7348	**84**	39315	**141**	54064
27	11450	**86**	1096	**142**	7355
40	3522	**93**	19664	**150**	16339
42	3296	**95**	68634	**154**	69878
49	31301	**102**	26846	**163**	19500
58	181613	**105**	11966		
60	3659	**111**	156177		

Table 2: Loss Values of Test Videos

Above loss table shows the MSE loss for each test video. Loss is not very bad at the beginning however it gets worse with each frame. When bounding box of one frame is missed, the rest of the results gets worse. Also, the highest loss here does not mean the worst prediction in my opinion. When GIFs are reviewed, despite loss is very high, intersection between bounding boxes are high. The loss is also effected highly from bounding boxes' size difference. However, in visualized tracking this is relatively less important issue. I found the MSE loss for test videos insufficient to represent the similarity. That's why the best/average/worst case prediction GIFs that are shared are not related to loss but to intersection of boxes.

The most common issue I have observed with tracking is that once track is lost, it is very hard (even impossible when it is out of where to look region) to recover it. This is the average case problem as I observed. (video 58 - average example) The validation and train phases are not continuous but it is done over a pair, that is why they are a bit more successful. Also the scaled image coordinates effect them positively. Another big issue is that estimating bounding boxes size and shape very poorly. In train dataset there are many examples with changing bounding box. Also motion size, zoom in / zoom out (video 95) effects this. I found it hard to learn for the network with only 1x1024 features. I think this is a small size of vector to estimate the motion.

Another issue is fast motion. Network is very insufficient to track the fast moving objects (video 119 - worst case, 93, 150, 163). This is the worst case scenario.

The problem with data is for some videos, ground truth bounding box is very small like in the case of video 27. The ground truth box only focuses the head of turtle, while the body also moves. This is hard to distinct especially when the surrounding region is also moving.

The network is good to track when the background is more steady and object is distinct and not moving very fast. (video 60 - the best case)

GIFs for worst best and average case is shared in `https://drive.google.com/drive/folders/1u3dgRFFr1MQRToHHJ_Its0jXREHZ1DxX?usp=sharing` . Red box shows the ground truth, white box shows the predicted bounding box.