Soccer Robot Perception
LabVision

Khan, Sarah[1] and Hashem, Mahmoud[2]

[1]s6srkhan@cs.uni-bonn.de, Matrikelnummer: 3279206, Bonn University
[2]s6mahash@cs.uni-bonn.de, Matrikelnummer: 3201329, Bonn University

Universität Bonn

Abstract. In this report, we present the implementation and training details of the deep learning vision system that was used to get The Best Humanoid Award of the RoboCup Humanoid League 2019 in Sydney, which is used to detect the position of balls, goal posts and robots as well as perform pixel-wise segmentation of the field and lines. We will demonstrate changes that helped in training the model, also show full model evaluation and shade the light over a few problems that we encountered.

1 Introduction

This report will illustrate NimbRoNet2 model (Rodriguez et al., 2019) implementation and training details. This model is the major component in the visual perception pipeline of the winning team in RoboCup Humanoid League 2019 in Sydney.

NimbRoNet2 is a unified deep convolutional neural network with an encoder-decoder architecture with two output heads to perform object detection for robots, soccer ball and goalposts, in addition to pixel-wise segmentation to recognize line segments and field boundaries in one forward pass.

First, the report will start with the preprocessing and loading of both detection and segmentation datasets, followed by our hyperparameters selection and reasons behind those choices. Then we will demonstrate model architecture, after that we will talk about the training cycle and how we performed alternative training between both datasets. Furthermore, we will show a few visualization analytics that helped severely in training and finally, we will report our metrics with a few model output results.

This architecture is depicted with a Flowgraph in Fig. 1.

2 Methods

2.1 Data Loading

We used the datasets provided by (*TeamNombRo*) which consisted of two sets, one for detection and one for segmentation. Datasets were divided into 70,15 and

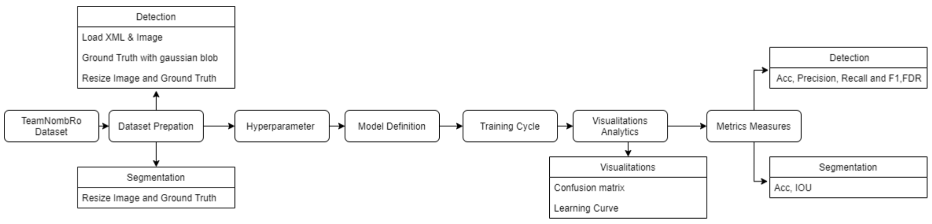

Fig. 1: Report Architecture

15 percent for training, validation, and testing respectively. We used a (*Custom DataLoader*) to load data in the desired manner.

For detection, we resized input images width and height to 640,480 respectively, using PyTorch transforms, because different input sizes have drastically varying gradients, especially when small images batched together, which affected training negatively and also we did not notice any noticeable deformation from that resizing, then we normalized it using normalize transform with 0.5 mean and standard deviation. Ground truth for detection training is a tensor of shape: classes * (1/4th width) * (1/4th height), where width and height are of an input image. Here, we include the probabilities of a particular class at the given pixel. For constructing ground truths for the detection task, we calculated the centers by obtaining coordinates of the bounding box from corresponding provided XML in raw data. As suggested in paper[1], for goalposts and robots we used bottom midpoints as the center. We scaled the coordinates of centers obtained as the output is 1/4th the size of the resized input. To perform that, center coordinates were scaled by resized output dimensions/original dimensions from XML. We visualize a few examples by plotting them along with corresponding input images to make sure that the obtained centers are correct. After obtaining the center we obtained the indices for a Gaussian blob by sampling a normal distribution with mean as the center and the variance as the radius of blob with the number of samples with a ratio 100:1 to the radius. Then we calculated one hot probability representation. Along with this we also create ground truth to calculate metrics, which contains the information of just the centers of the classes. We fine-tune radius to minimize any huge overlapping between classes as this will lead to shifted means of classes for detection training. An example of an input image and its corresponding target image is depicted in Fig. 2 and Fig. 3.

For segmentation, we resized input images to the same dimensions mentioned in detection followed by the same normalization process mentioned above. For ground truth, we resize this image to have dimensions 1/4th of the input image. We obtain float values in range 0 to 1 rather than actual labels. So, to obtain labels, we multiplied by the obtained target by 255. Few images instead of having only 3 classes consisting of background, field and field lines also included a fourth class for balls. We obtained the indices having the label for balls and mark it with a label for a field. Hence, for an input image of dimension 640*480, we had

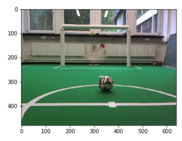
Fig. 2: Object detection Input Image

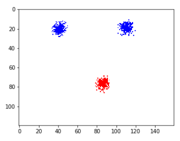
Fig. 3: Object detection Ground Truth

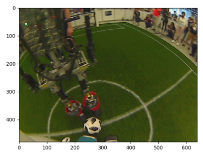
Fig. 4: Segmentation Input Image

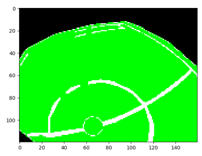
Fig. 5: Segmentation Ground Truth

a corresponding target image of 160*120 which consisted of labels for the three classes for each pixel. An example of an input image and its corresponding target image is depicted in Fig. 4 and Fig. 5.

2.2 Hyperparameters

Hyperparameters play a crucial rule in the quality of the trained model that's why we explored many parameters including learning rate, batch sizes, optimizers, loss functions, and total variation loss weights .

For choosing the appropriate learning rate, we started by small value 0.001 in fear of quick divergence but that resulted in a very slow training process with 60-70 epochs noticeable increase in model quality and hence we increased the learning rate to be 0.01 which was appropriate to balance between faster training process and risk to stuck in local minima. we investigated also 0.1 value which results in an underfitted model, also we noticed that a progressive decrease in the learning rate as training goes leads to faster convergence, we experience that by training with 0.1 value for few epochs and then decrease it to 0.01, but as we used Adam optimizer to leverage the advantage of adaptive learning rate for individual parameters offered by it, we noticed that just starting learning rate with 0.01 was sufficient.

Batch size above 32 was resulting in GPU memory being full and hence, only allowing us to use CPU for training. But even, with batch sizes in the range of 25-30 resulted in poor generalization and slower convergence. Trying with batch size in the range of 10-15 although the convergence was faster, we got poor generalization over the test dataset. This happened because the model started learning towards a suboptimal minimum without having seen a variety of data in the training dataset. Hence, for our training purposes, we fixed batch size as 20.

We kept Rodriguez et al. (2019) loss functions, mean squared error loss along with total variation loss for object detection as it has huge error for drastic mistakes but low error for small ones. This property is advantageous in the detection task because we don't want to penalize the model for detecting objects with a slight difference in detected centers. For the segmentation task, instead of using softmax for raw outputs and then using negative log-likelihood loss, we have used cross-entropy loss which is equivalent to them, as shown in (*Explaination on using NLLLoss and Cross Entropy*), along with total variation loss also which generally helps in removing the rough texture in the output, resulting in the generation of smoother results. We want to have a total variation on the same scale as the other losses so that it does not dominate in training. So we fine-tuned the weight for total variation loss for segmentation to be 0.00001 and for detection 0.000001.

2.3 Model

We have used the model architecture of NimbRoNet2 which is an encoder-decoder architecture similar to pixel-wise segmentation models like (Badrinarayanan et al., 2017), and (Ding et al., 2019). The model uses convolutional layers of pre-trained ResNet18 as an encoder to minimize the need for a large dataset, which requires a huge annotation effort. Transpose-convolutional layers are used for up-sampling the multiple level representations extracted from ResNet which act as a decoder.

Instead of a convolutional layer, the model used a location-dependent convolution in the last layer to get location-dependent features and a shared learnable bias between both output heads was used to reduce the number of the model parameters. The model output consists of two heads; one for object detection, and the other for pixel-wise segmentation. The detection head gives the location of the ball, robot, and goalposts. The segmentation head is for line, field detection.

The background is detected as a separate class in segmentation while as a lack of confidence in all classes in detection. The NimbRoNet2 visual perception architecture is illustrated in Fig. 6.

2.4 Training Cycle

In one epoch we train detection and segmentation alternatively over the complete dataset. We achieved this by obtaining a batch from the detection dataset, calculate detection losses of that batch and then backpropagate loss through the network for the same, followed by the same behavior for batch for segmentation. Since training of object detection was taking more time as well as the training dataset size of object detection was significantly larger than segmentation, we increased the ratio of training for the detection task to that of the segmentation. We found improved results when we train 4 batches of detection followed with a batch of segmentation and in case that any of the datasets finished before the

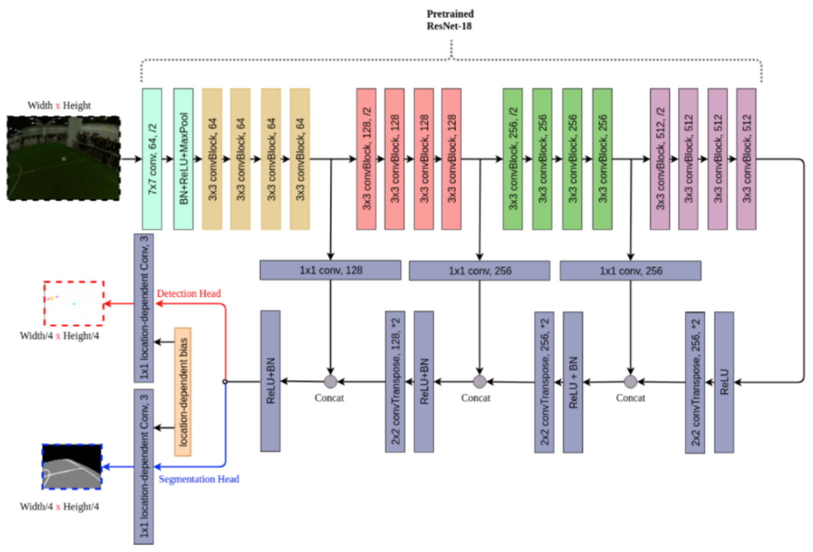

Fig. 6: NimbRoNet2 architecture [3]

other, we move to the next epoch with giving fetched the left out batches first to make sure that every batch participates in training.

For termination, we have used early stopping, in which we use the averaged accuracy of both tasks on validation datasets, so we terminate the training, either after a patience number of epochs without improvement in validation accuracy, or when we complete all epochs of training. We also keep on saving model with its current state and losses whenever we get better accuracy on the validation.

For object detection, we sum mean squared error loss along with total variation loss and backpropagate them. We did not take the maximum of the two losses as then it would mean that for one batch it would either learn features or learn denoising the output image.

For the segmentation task, cross-entropy loss with total variation loss. (*Explaination on using NLLLoss and Cross Entropy*). Total variation loss for segmentation does not include the class of field lines as these lines are few pixels thick and smoothening them could result in complete pixels for field lines in segmented images.

3 Results

3.1 Visualisation Analytics

We show a sample of the results from the model after training in Fig. 7 and also the effect of lighting condition, we have noticed that as the brightness increases detection accuracy increases, on the other hand, segmentation accuracy decreases and vice versa which is depicted in Fig. 8.

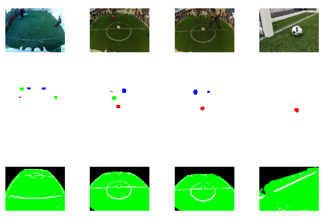

Fig. 7: Model results. Upper row: input images. Middle row: the output of the detection head with balls (red), goal posts (blue), and robots (green). Bottom row: the output of the segmentation head with lines (white), field (light green), and background (black)

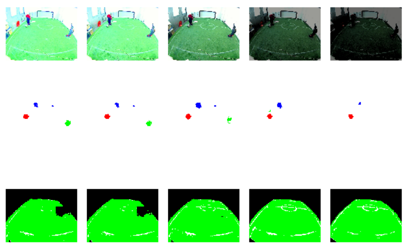

Fig. 8: Model result with different illuminations. far left column: brightest input. left column: brighter input. middle left column: original input. right column: dim input. far right column: dimmest input.

We provide a few analytics that gave us some helpful insights in training the model First, Fig. 9,Fig. 10 shows the learning curve for the Detection and Segmentation training and validation respectively, we use it to implement early stopping criteria that were useful in preventing overfitting.

Then, confusion matrices for both output are shown in Fig. 11,Fig. 12 which helped us determine the correlation patterns in the data that captured by the model and it gave us an indication of certain classes shortages in datasets like field lines in segmentation and goal pillars in case of detection.

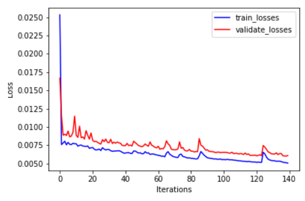

Fig. 9: Detection Learning Curve

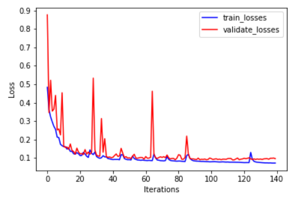

Fig. 10: Segmentation Learning Curve

3.2 Metrics for Object Detection

In detection, we provide both per class and overall results in Table 2. We could not achieve the same metrics reported by NimbRoNet2 except for precision and false detection rate were we achieved comparable results. We suspect that this behavior is because of the big difference in the representation of each class in the

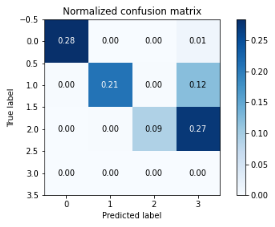

Fig. 11: Detection Confusion Matrix. ball:0, goal pillar:1, robot:2 and background:3

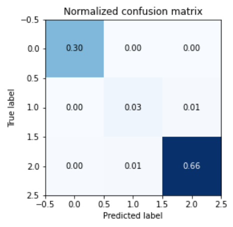

Fig. 12: Segmentation Confusion Matrix. background:0, field lines:1 and field:2

dataset and it can be overcome by balancing the numbers training passes that each class performs during the training. we tried to improve the result by more detection to segmentation training ratio, but unfortunately, that just slightly improve results.

To calculate those metrics, we needed to calculate True Positive (TP), False Negative (FN) and False Positive (FP) for each class.For that, we need to find possible objects' centers in predicted output. Once those centers are calculated, we then used a (*KDTree*) for alignment between the centers of objects in predicted and centers in the ground truth, then we filtered some of the predicted centers using a threshold on distances, which leave us with both centers aligned based on closest distances in which we can calculate TP, FN and FP from easily.

We still need to get possible predicted objects centers from predicted detection maps, which are performed in that order. First, we generate a colored detection image from predictions by assigning each class to a distinct color. Then we perform a few (*Morphological Transformations*) on that colored image, which is opening followed by closing in order to remove noises in the image and fill any holes inside blobs in the image, which significantly increase the quality of centers calculated by reducing the generation of multiple centers that belong to the same object. After that we pad the processed image with a white border to avoid combining contours for objects on the edges of the image then we perform (*Find Contours Algorithm*) for each class channel to prevent connecting contours from different classes. Finally, we calculate the centers by averaging the contours point.

Once we have TP, FP, and FN for each class, calculations are straight forward. For per class metric, Eq. (1),Eq. (2),Eq. (3),Eq. (4) and Eq. (5) explain the calculations, Where $TP_i + FN_i + FP_i$ are $TP + FN + FP$ for class i respectively.

$$Accuracy_i = \frac{TP_i}{TP_i + FN_i + FP_i} \tag{1}$$

$$Recall_i = \frac{TP_i}{TP_i + FN_i} \tag{2}$$

$$Precision_i = \frac{TP_i}{TP_i + FP_i} \tag{3}$$

$$FDR_i = \frac{FN_i + FP_i}{TP_i + FN_i + FP_i} \tag{4}$$

$$Accuracy = \frac{1}{C}\sum_{i=1}^{C} Accuracy_i, \tag{5}$$

For overall metrics we average all of the metrics over all C classes Eq. (6).

$$Metric = \frac{1}{C}\sum_{i=1}^{C} Metric_i \tag{6}$$

3.3 Metrics for Segmentation

In segmentation, to grasp how the model is performing regarding each class, we provide both per class and overall results in Table 3, we have achieved better accuracies in all classes than reported by NimbRoNet2, especially in lines with more than 10 percent in differences and nearly same IOUs in all classes except in lines IOU, which is lower by 10 percent than what was reported.

We got most metric definitions fromMinaee et al. (2020). For each class accuracy, We used **Pixel Accuracy (PA)** per class which is simply the ratio of class pixels correctly classified, divided by the total number of pixels per class. For C classes (foreground classes and the background) pixel accuracy for class i is defined as Eq. (7).

$$PA(i) = \frac{p_{ii}}{\sum_{i=1}^{C} p_{ij} + \sum_{i=1}^{C} p_{ji} - p_{ii}} \tag{7}$$

Where p_{ij} is the number of pixels of class i predicted as belonging to class j.

For overall accuracy, We used **Mean Pixel Accuracy (MPA)** which is an average of Pixel Accuracy over the total number of classes is defined as Eq. (8)

$$MPA = \frac{1}{C}\sum_{i=1}^{C} PA(i) \tag{8}$$

Intersection over Union (IoU) is defined as the number of the pixels in area of intersection between the predicted and the ground truth, divided by the number of the pixels in area of union between the predicted and the ground truth as in Eq. (9).

$$IOU(X,Y) = \frac{\mid X \cap Y \mid}{\mid X \cup Y \mid} \tag{9}$$

Where X, Y are the predicted and ground truth respectively.

For each class IoU, we use **IOU** with X_i, Y_i, which are the predicted and ground truth pixels with class i respectively.

For overall accuracy, We used **Mean-IOU**c, which is defined as the average IoU over all classes as in Eq. (10).

$$Mean - IOU = \frac{1}{C}\sum_{i=1}^{C} IOU(X_i, Y_i) \tag{10}$$

Table 1: Model Results

Table 2: Results of the detection branch of our visual perception network.

Type	F1	Accuracy	Recall	Precision	FDR
Ball	0.9752	0.9538	0.9634	0.9899	0.0100
Goal	0.3856	0.2449	0.2481	0.9503	0.0494
Robot	0.7624	0.6244	0.6329	0.9784	0.0214
Total	0.7077	0.6077	0.6148	0.9729	0.0269

Table 3: Results of the semantic segmentation of our visual perception network.

Type	Accuracy	IoU
Field	0.9813	0.9715
Lines	0.9846	0.6563
Background	0.9941	0.9816
Total	0.9867	0.8698

4 Conclusion

In conclusion, we have illustrated a detailed guideline for the implementation and training of NimbRoNet2. We have conducted a wide range investigation in order to fine-tune the hyperparameters and understand their direct effect on the model quality as shown in sec 2.2. We have also shown some useful tricks to increase the convergence rate of the model as shown in sec 2.4 and shared some of the problems we have encountered and handled it as in sec 3.2. we hope to further investigate in different models architecture as in Huang et al. (2017) which tackle the vanishing gradients problem and is also highly parameter efficient which is essential due to robots limited resources.

References

Badrinarayanan, Vijay et al. (Dec. 2017). "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: DOI: 10.17863/CAM.17966.

Chilamkurthy, Sasank. *Custom DataLoader.* Tutorial on how to write custom dataloader in pytorch, https://pytorch.org/tutorials/beginner/data_loading_tutorial.html.

Ding, Yi et al. (July 2019). "A Stacked Multi-Connection Simple Reducing Net for Brain Tumor Segmentation". In: *IEEE Access* PP, pp. 1–1. DOI: 10.1109/ACCESS.2019.2926448.

Huang, Gao et al. (July 2017). "Densely Connected Convolutional Networks". In: DOI: 10.1109/CVPR.2017.243.

Minaee, Shervin et al. (Jan. 2020). *Image Segmentation Using Deep Learning: A Survey.*

Morphological Transformations. Morphological transformations are some simple operations performed to get a more useful image for tasks of detection and recognition, https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html.

Opencv. *Find Contours Algorithm.* Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity., https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html.

ptrblck$_{d}$e. *Explaination on using NLLLoss and Cross Entropy.* Explaination on using NLLLoss and Cross Entropy, https://discuss.pytorch.org/t/does-nllloss-handle-log-softmax-and-softmax-in-the-same-way/8835/2.

Rodriguez, Diego et al. (Dec. 2019). "RoboCup 2019 AdultSize Winner NimbRo: Deep Learning Perception, In-Walk Kick, Push Recovery, and Team Play Capabilities". In: pp. 631–645. ISBN: 978-3-030-35698-9. DOI: `10.1007/978-3-030-35699-6_51`.

Scipy. *KDTree*. kd-tree for quick nearest-neighbor lookup, `https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.spatial.KDTree.html`.

TeamNombRo. *TeamNombRo*. Datasets for object detection and pixel-wise segmentation, university of bonn, `http://bigcuda5.informatik.uni-bonn.de:8686/`.