IE643 Programming Challenge Report

Shivprasad Kathane 180110076

I first implemented U-Net on the Challenge Dataset but it didn't yield any good results at all. Some of the images seemed to contain camouflaged objects. So, it's not simple segmentation - a subtler approach is required which focuses on the boundary between the camouflaged object and its surroundings. I searched in the literature for what researchers have tried out. This is where I came across this paper [Boundary-Aware Segmentation Network for Mobile and Web Applications], which proposed Boundary-Aware Segmentation Network (BASNet) capable of highly-accurate image segmentation. The architecture as shown below consists of 2 modules: corresponding to prediction and refinement which is optimised using a hybrid loss function.

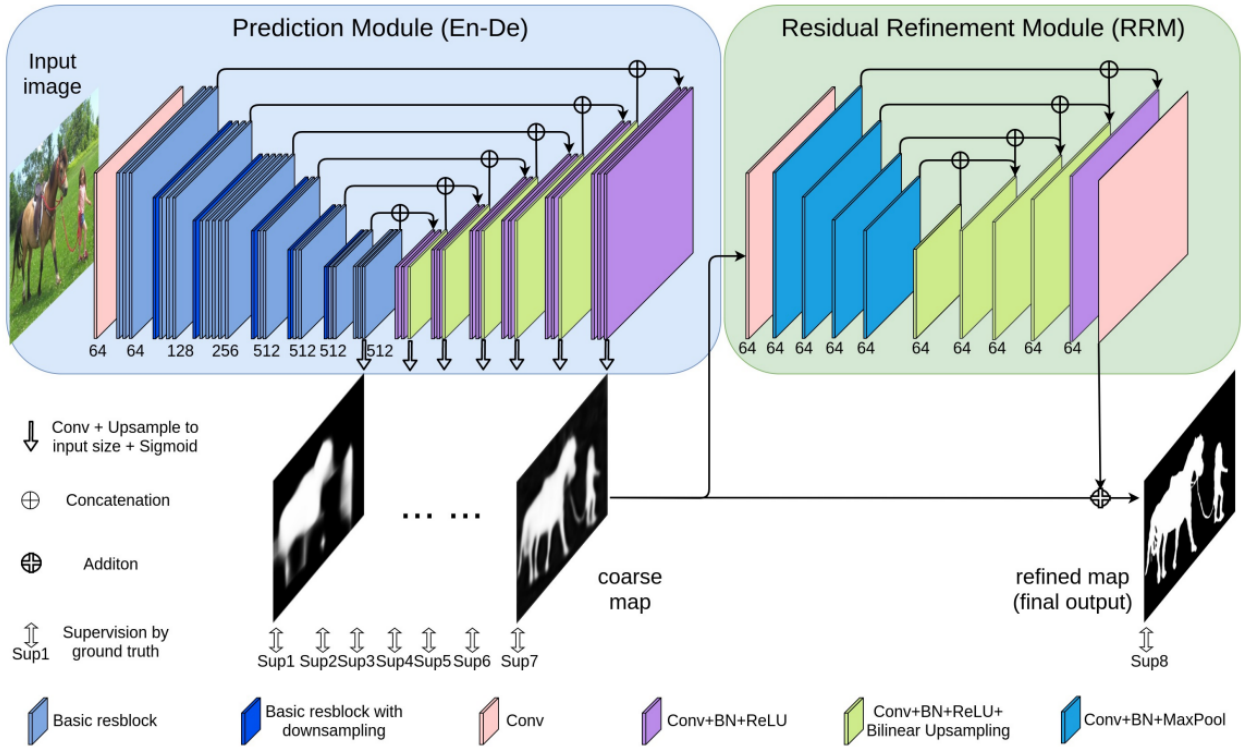


Fig. 2. Architecture of the proposed boundary-aware segmentation network: BASNet. See § 3 for details.

The prediction module consists of pre-trained layers of ResNet-34 followed by encoding and decoding blocks with concatenation as in U-Net while the residual refinement module is a simple U-Net like architecture of convolutional layers with the purpose to refine (sharpen boundary) the segmentation mask predicted by the earlier module. The code for this network is available in pytorch at https://github.com/xuebinqin/BASNet.

Taking inspiration from this, I have designed a similar architecture in keras. I have used pre-trained ResNet-50 (which is supposed to yield more accurate results than ResNet-34 ref: https://viso.ai/deep-learning/resnet-residual-neural-network/) essentially for feature extraction. It consists of a simple convolution block followed by 4 encoder blocks (each block having 3 convolutions opposed to 2 in ResNet-34). Then, I have added a custom encoder (residual) block followed by the bottleneck and 5 decoder blocks, with each decoder block taking inputs from both the previous block output and the corresponding encoder block output (concatenation). Batch-Normalisation, ReLU, Maxpooling follows the convolution layer as applicable. The purpose of using an encoder-decoder architecture with concatenation remains the same as the traditional way to simultaneously capture high-level global contexts and low-level details. The network yields output after each bottleneck or decoder block with upsampling using bilinear interpolation to map to the output space. This is done so that each layer output is supervised with the ground truth mask. The final convolution output after the last decoder, is fed into the refinement network. This network is like a simple U-Net with input convolution followed by 4 encoding, bottleneck and 4 decoding layers (each with single convolution) and finally output convolution which is the final model output (predicted mask) - desired to be a refined version of the prediction model output. More details available in my code along with the model summary and architecture flow chart. This model had 450M+ trainable parameters. I realised after some training that this model is very complex, computation is slow and the size of the model file was approx. 5 GB which made it difficult to save/download/upload. So I made it simpler by removing the extra encoder and corresponding decoder block and trained it again. This has been done in the CODE2 file. I made the model more simpler by taking output from ResNet before its last block and removing a decoder block. This model has 46M+ trainable parameters and is coded in the CODE1 file.

Data Pre-Processing: As each image is of different size, some data preprocessing is required so that the same shape is passed as input to the model (ResNet input).

- Each image is resized to square shape of 128 x 128 across 3 channels
- The pixel values for each image are normalised into 0-1 range from 0-255

Data Augmentation - In order to increase the dataset size for training:

- A copy of each image is added to the dataset by flipping it randomly either horizontally or vertically or combination of both.
- Another copy of each image is added to the dataset by rotating it by an angle randomly chosen between 1 and 359 degrees.
- Care is taken to apply the same transformations on the corresponding masks.

This essentially triples the dataset size for both training and validation purposes.

Loss Function: A hybrid loss function as a sum of 3 loss terms is designed.

1) The SSIM or Structural Similarity Index Measure is a patch-level measure of the similarity between the predicted image and the true image. It takes the local neighbourhood of each pixel into consideration and assigns higher weights to pixels around the boundary. So dissimilarity loss = 1-SSIM is taken as one term.
2) The IoU or Intersection Over Union is a map-level measure which emphasises more on the large foreground regions. Jaccard Loss = 1-IoU is taken as the loss.
3) The BCE or Binary Cross-Entropy Loss is computed pixel-wise. It does not consider the labels of the neighbourhood and weights both the foreground and background pixels equally. This helps with the convergence on all pixels and guarantees a relatively good local optima. This forms the third term.

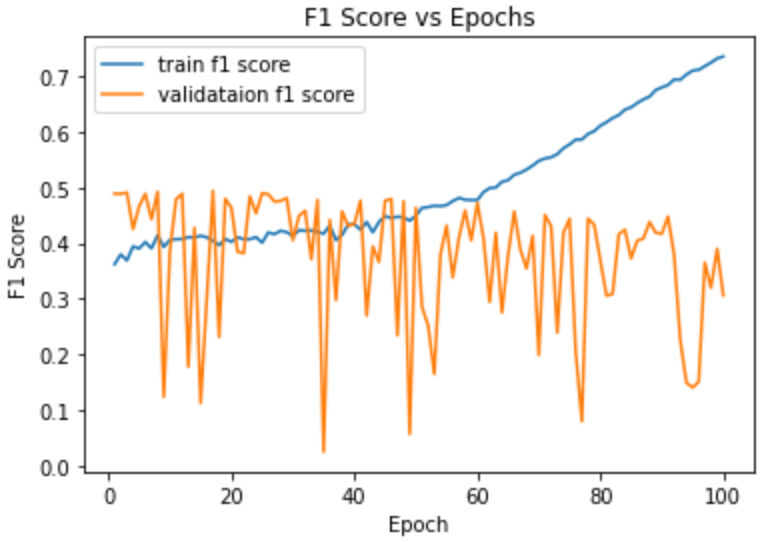
Hence, the aggregation of their individual relevance to the problem is expected to help in optimising the weights so that better segmentation happens at all the 3 levels.

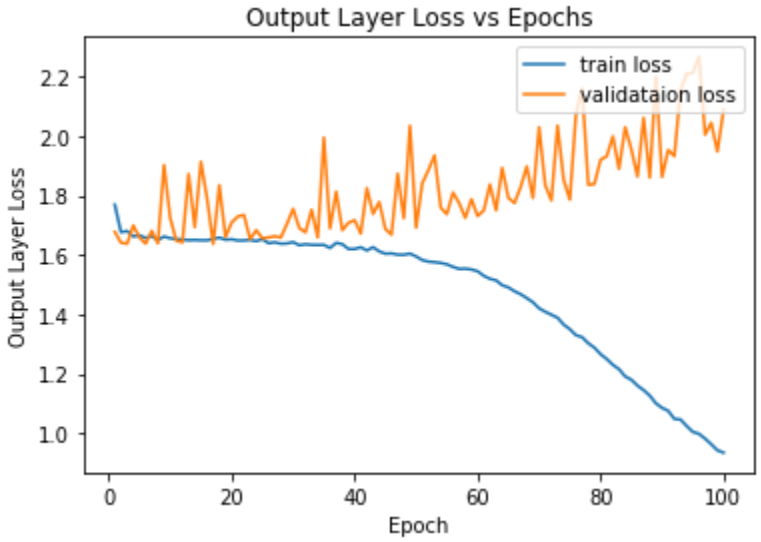
Adam Optimisation with the default settings is used for training. This means constant learning rate = 0.001, beta_1 = 0.9, beta_2 = 0.999, and epsilon = 1e-07. The model is trained for a constant number of iterations (no early stopping). 50 epochs for the initial complex model. The most simple one was trained for 100 epochs before saving, loading and retraining for 50 more epochs. At each epoch, the model was trained on the training set with the loss and performance measured on both the training and validation set. The batch size was 8 for the complex model while 10 for the simplest one. Some hyperparameters and their values include initial no of filters = 64, filter size = 3, stride = 1, conv activation = ReLU, padding = ‘same’. Hyperparameter tuning wasn’t carried out due to ambiguity regarding the training convergence and high time consumption.

F1 Score = Dice Coefficient = 2TP/(2TP+FP+FN) = 2PR/(P+R) is taken as the main performance metric although accuracy is also recorded. It takes into account both precision (P) and recall (R) so that predictions for both the classes (object/background) are equally accounted for during measurement. Further, as F1 is similar to IoU, the hybrid loss is expected to maximise F1 as training proceeds.

The simplest model which was trained for 150 epochs in total has been saved and uploaded for submission. The final loss and metrics for the model output were:
Loss (train) = 0.4933 (least) while Loss (val) = 1.9822 (least = 1.6369)
F1 score (train) = 0.8781 (max) while F1 score (val) = 0.3925 (max = 0.4941)

The corresponding plots showcasing the variation in them up to 100 epochs are:





From these plots, it seems that training is still in progress as the training loss keeps on decreasing and the training F1 keeps increasing. And also, as it hasn't learnt the optimum weights (which is evident from the predicted masks), it is far away from generalisation, so there isn't desired performance on the validation set (fluctuating but in fact decreasing F1 and increasing loss is observed as net effect which is undesired). This might be expected because the authors in the paper had reached good performance after training for 110,000 iterations which is far greater than 100 epochs.