Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Bonn-Aachen
International Center for
Information Technology

Master's Thesis

Sonar Patch Matching via Deep Learning

Arka Mallick

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr. Paul G. Plöger
Prof. Dr. Gerhard K. Kraetzschmar
Dr. Matias Valdenegro-torro

November 2018

Contents

1

Evaluation and results

1.1 Densenet

In Densenet each layer connects to every layer in a feed-forward fashion. With the basic idea to enhance the feature propagation, each layer of Densenet blocks takes the feature-maps of the previous stages as input.

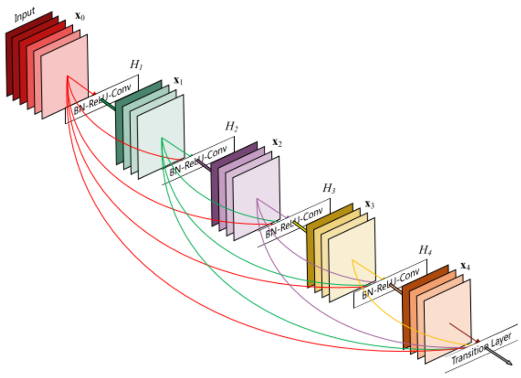


Figure 1.1: Densenet structure.

1.1.1 Best hyper-parameters search spaces

Hyper-parameters are those parameters whose values are set before the training, unlike other parameters whose values are learned during the training. For example learning rate, batch size. [1]

Parameters of Densenet

- **Growth rate**: Number of filters to add per dense block. Growth rate regulates how much information is contributed by each layers to the global state. Global state is the collective knowledge of the network, that is the state of the previous layers are flown into each layer in form of feature-maps, which is considered as global state. Each layer adds k more feature-maps to the current state, when growth rate is k.
- **Nb_filter**: initial number of filters. -1 indicates initial number of filters will default to 2 * growth_rate.
- **Nb_layers_per_block**: number of layers in each dense block. Can be a -1, positive integer or a list. If -1, calculates nb_layer_per_block from the network depth. If positive integer, a set number of layers per dense block. If list, nb_layer is used as provided. Note that list size must be nb_dense_block.
- **Depth**: number or layers in the DenseNet.
- **Nb_dense_block**: number of dense blocks to add to end.
- **Bottleneck Layers**: To improve computation efficiency a bottleneck layer with 1x1 convolution is introduced before each 3x3 convolution layers.
- **Compression**: Reduces the feature maps in transition layers and makes the model more compact and computationally efficient.

Other hyper-parameters

Apart from the densenet there are other general hyper-parameters such as learning rate, batch size for training, optimizer.

Grid search strategy

Since there are lot of parameters, practically, infinite test cases might be designed. To keep the grid search focused and less computationally expensive it makes sense to first search for coarser grid of parameters rather than very fine ones. When and if a bracket of parameters are shortlisted which works better than others, the finer parameter search will be performed only specific to those range of parameters and not the whole grid.

1.1.2 Analysis

Coarse grid search on Densenet hyper-parameters

For the estimation of the best performing parameters of Densenet for the branches of the Siamese, first a coarse grid search is been performed with

- Layers per block are chosen among 2,3,4,5,6. For single dense block evaluation goes up-to 12 layers (per block).
- Each network has been evaluated for growth rates of 12,18,24,30.
- Different dense block sizes of 1,2,3,4.
- The basic idea here is to narrow down the possible network sizes from the Densenet parameters perspective. There are other parameters but number of dense block, growth rate and layers per block are three main parameters which controls the architecture/size of the network the most.
- The parameters compression/reduction and bottleneck are set 0.5 and False respectively. For more fine grained analysis the compression and bottleneck parameters might be evaluated.
- nb_filter values are fixed at 16 for this test
- The parameter classes are set to 2, where class 1 for matching patches and 0 for not-matching patches.

- 96,96 is the input image dimensions. And input is two channel. So depending on local setting of the keras, channel-first or channel-last suitable input_shape is chosen automatically as 2,96,96 or 96,96,2 respectively.
- The learning rate used for the test was is 0.07 and Adadelta as optimizer. Best performing combination from Densenet Siamese analysis.
- Dropout for Densenet used as 0.2 to handle regularization.
- Epochs are different for different architectures to ensure that the networks are able to train decently
- Some of this values needs to be further evaluated as well, how ever current values were obtained using some of manual tuning and assumed to be a decent starting point.
- Flatten is used as pooling at the end of the Densenet, it is introduced by this work which replaces the Globalaveragepooling step with a flatten. This causes increase in parameters overall though.
- Binary_crossentropy loss function with sigmoid activation function used for the binary classification, this final layer acts as the binary classifier.
- In all the cases the networks are being trained from scratch. The weights value None ensures that no previously trained weights are used.

Coarse grid search parameters summary

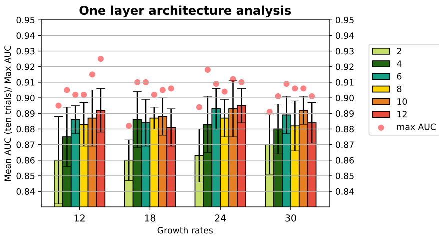
Fixed hyper-parameters

- Nb_filter: 16
- Subsample initial block: True
- Weights : None
- Dropout rate : 0.2
- Include top : True

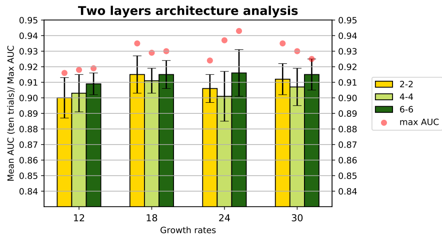
- Compression : 0.5
- Bottleneck : False
- Transition pooling : max

Varying hyper-parameters

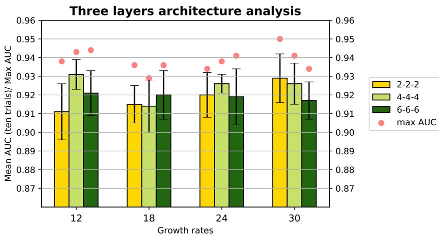
- **Nb_layers_per_block**:
  - **One dense block architecture (nb_dense_block=1):**
    '2', '4', '6', '8', '10', '12'
  - **Two dense block architecture (nb_dense_block=2):**
    '2-2', '4-4', '6-6'
  - **Three dense block architecture (nb_dense_block=3):**
    '2-2-2', '4-4-4', '6-6-6'
  - **Four dense block architecture (nb_dense_block=4):**
    '2-2-2-2', '4-4-4-4', '6-6-6-6'
- **Growth rate**:
  - Thin layers: 12, 18
  - Thick layers: 24, 30
- **Pooling**:
  - Flatten
  - Average (avg)



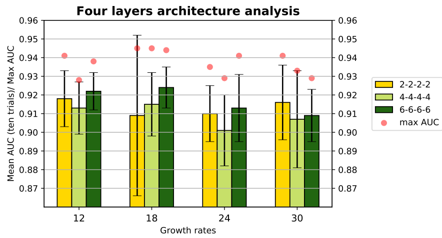
(a) Single layer



(b) Two layers



(c) Three layers



(d) Four layers

Figure 1.2: Densenet two-channel architecture analysis

Architecture analysis

Conclusion

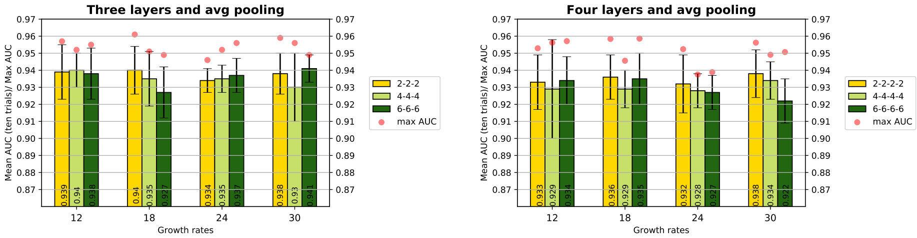
- It is clear that 4 blocks and 3 blocks network is performing better than the 1 and 2 blocks network

- Original paper also used 3 and 4 blocks densenet, so shall we.

Pooling avg vs flatten

The type of pooling used at the end of the network also determines the size of the total parameter size and also affects the generalization of the data. The original paper has used global average pooling. For the avg pooling comparison the 3 layer and 4 layer Densenet blocks have been compared. The results are as follows. It is observed that between the three layer and four layers Densenet the three layers one is performing better. Overall the avg pooling is resulting in smaller total network parameters also the better results.

For a comparitive analysis between the flatten and average pooling the mean AUCs are compared for each of the growth rates and for three layers Densenet(



(a) Three layers avg pooling (b) Four layers avg pooling

Figure 1.3: Densenet two-channel avg pooling analysis

2-2-2, 4-4-4, 6-6-6). Results are displayed in the figure below.

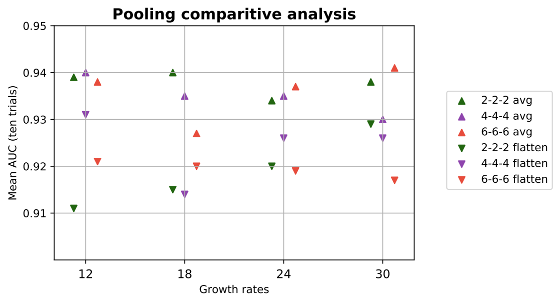


Figure 1.4: Average vs flatten pooling

Here, the triangles up represents the mean AUC obtained using average pooling for each of the test cases. And triangle down represents flatten pooling. While the mean AUC for each of the three layers Densenet architectures have been grouped according to four different growth rates. So the x axis of the graph shows growth rates 12, 18, 24 and 30. The y axis represents the mean AUC obtained from 10 trials. Each time the network, with the same settings, were trained from the scratch. From figure 1.4 it is clearly seen that the average pooling produces better results than using flatten, in all the cases.

Nb_filter analysis

Initial number of filters. 8,16,32,64 are being evaluated here. Also a comparison between mean AUCs obtained from growth rate 18 and 30 is done under this analysis. Ten evaluations of each test cases has been done.

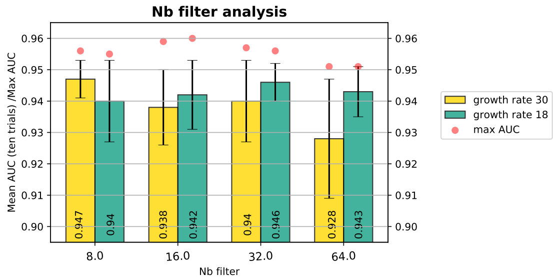


Figure 1.5: Nb filter size analysis analysis

Conclusion With change in nb filter size the mean AUC varies alot for higher growth rate such as 30, for growth rate 18 it does not vary so much. For growth rate 30 the nb filter 8 has the best mean AUC. For growth rate 18 the nb filter 32 has the best mean AUC. Overall growth rate 30 with nb filter 8 and growth rate 18 with nb filter 32 are the best combinations.

Reduction and bottleneck analysis

This analysis is for evaluating the effect of different reduction rates and the effect of bottleneck. So the mean AUC was recorded for 10 trials for each of the reduction values 0, 0.2, 0.3, 0.5, 0.7. This is a rather coarser search space. But each of them were also evaluated with bottleneck, the effect of varying values of reduction and with/without bottleneck has been evaluated. The results are displayed in the figure below.

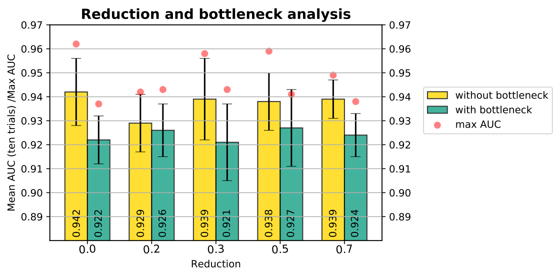


Figure 1.6: Reduction and bottleneck analysis

Conclusion

- The effect of the bottleneck layer is rather limiting the generalization of the network. So it seems like without bottleneck should be used for future evaluations.

- The performance without reduction is best as expected, how ever main purpose of the reduction is to decrease the number of total parameters. So in that sense, mean AUC obtained of reduction 0.3, 0.5 and 0.7 are all very good even though the size of the total parameters is much lower. So values around 0.7 will be used for the final grid search. In the original implementation 0.5 is the default value used.

- TODO: the value for 0.2 is rather unexpected. The value was expected to be lesser than without reduction and with very high reduction.

Total parameters analysis

- For the comparison 2, 2-2, 2-2-2, 2-2-2-2 blocks parameter sizes are compared below, all recorded for growth rate 18.

- FLATTEN VS AVG POOLING and effect

- Reduction effect

Standard deviation across blocks

TODO

Dropout analysis

Densenet dropout: 10 evaluations each

It is observed that the 0.2 dropout configuration has obtained the highest mean AUC. The other values with lesser dropouts or greater dropouts are all gradually decreasing as they go further from the peak (0.2). With exception of the mean AUC obtained with 0.7 dropouts.

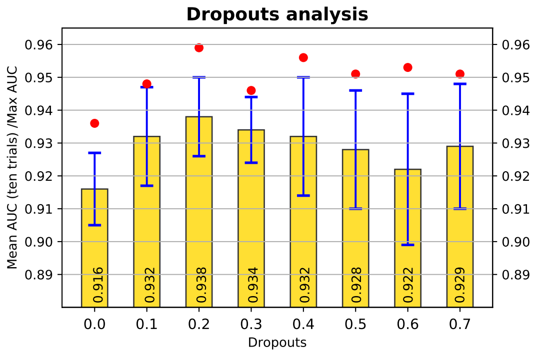


Figure 1.7: Dropouts analysis

Batch size analysis

If the batch size is too low then it takes more time and after a certain size it does not train well too.
If the batch size is very big then it may train faster but they generalize lesser as they tend to converge to sharp minimizers of the training function. TODO add source (https://arxiv.org/abs/1609.04836)

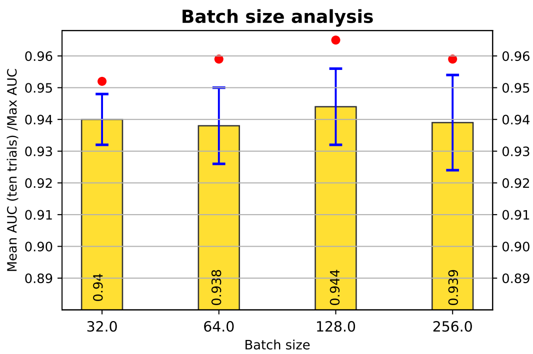


Figure 1.8: Batch size analysis

Conclusion From the figure 1.8 it is concluded that the batch size of 128 works the best.

Learning rate and optimizer analysis

For this analysis the Adadelta optimizer is used only. This is based on the intuition that was formed during the Densenet Siamese evaluation.

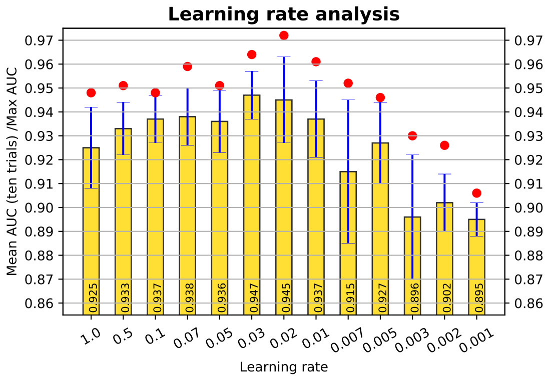


Figure 1.9: Learning rate analysis

Conclusion From figure 1.9 it is observed that the mean AUC with the learning rate 0.03 is slightly higher than the others. While one of the evaluation with 0.02 learning rate has obtained the maximum AUC of 0.972. So the best learning rate is considered to lie around the 0.02-0.03 region.

Hyper-parameters for final grid search

- Growth rate, in the original paper, authors have mentioned that without bottleneck and compression the general trend is to use as high as possible growth rate. For the Imagenet, they have used growth rate upto 40. For all their experiments they have evaluated growth rates from 12 to 40. Since we do not have so much of data, we will evaluate the finer grid search with growth rates of 12(thin), 18 and 30(thick).

- Layers per block 2-2-2 it has very consistant performance in terms of mean AUC and also able to score high max AUC. It could have been possible to do architecture searches for 3 dense block architectures with layers per block close to 2-2-2, for example 2-3-3 etc. But then the search grid will be very big.

Gr.	Metrics	Layers per block					
		2-2-2		4-4-4		6-6-6	
		R=0	R=0.5	R=0	R=0.5	R=0	R=0.5
12	Mean AUC	0.95	0.944	0.95	0.95	0.947	0.945
	Std	0.011	0.015	0.009	0.01	0.008	0.008
	Max AUC	0.97	0.97	0.963	0.965	0.963	0.955
	Total Parameters	55,529	30,163	159,473	87,629	317,561	176,535
18	Mean AUC	0.952	0.955	0.951	0.944	0.948	0.938
	Std	0.008	0.009	0.005	0.011	0.006	0.014
	Max AUC	0.967	0.966	0.956	0.963	0.956	0.955
	Total Parameters	96,785	51,430	308,369	168,671	640,481	355,860
30	Mean AUC	0.943	0.948	0.943	0.944	0.932	0.941
	Std	0.008	0.008	0.01	0.013	0.015	0.011
	Max AUC	0.959	0.964	0.96	0.962	0.948	0.953
	Total Parameters	160,001	82,162	650,873	355,949	1,473,665	822,276

Table 1.1: Final grid search results

- Bottleneck no
- Reduction 0 and 0.5
- Nb filter for 12, 18 growth rates use 32 and for growth rate 30 use 8.
- Dropout 0.2
- Adadelta with 0.03 learning rate
- Batch size 128

The result is displayed in the table 1.1. Here the multi dimension search space and associated results are displayed. Three architectures 2-2-2, 4-4-4 and 6-6-6 are evaluated for all three growth rates 12, 18, 30 and also for Reduction 0.5 and without Reduction. In the table 1.1 the growth rate is displayed as Gr. and Reduction is displayed as R for space constraint.

Conclusion

- The best result obtained has mean AUC of 0.955. This is with reduction 0.5, 2-2-2 layers per block and growth rate of 18. Normally it is observed that the 2-2-2 performance is very similar to that of 4-4-4, in fact slightly better. The performance of 6-6-6 is bit worse than the other too.

- Though because of reduction the auc is observed to be slightly lower some times, some times it is higher than the without reduction result. But the size of the total parameters of the network with Reduction(R)=0.5 is always close to half size of the equivalent network without Reduction. So that is always beneficial as it is less computationally expensive.

- In Matias's work it was also found that the simple two channel network better than the siamese network. Using Densenet it is also seen to be the truth.

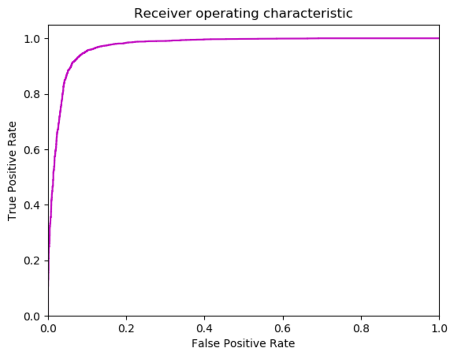


Figure 1.10: AUC curve Overall best result in both densenet architectures 0.973 AUC (Single run)

1.2 Comparitive analysis

So the three network structure that were used, will be compared in this section, in terms of AUC value obtained. Time of execution, total parameters also by using the uncertainty calculated from the Monte Carlo drop out calculations. All our final models were trained with dropouts, as that was the best parameters set up for the network.

Network	Mean AUC	Std	Max AUC	Total params
Densenet two channel	0.955	0.009	0.966	51,430
Densenet Siamese	0.921	0.016	0.95	16,725,485
Contrastive loss	0.944	0.007	0.951	3,281,840

Table 1.2: Comparative analysis on the AUC and total number of parameters in the best performing networks.

1.2.1 AUC comparison

Densenet two channel has highest mean AUC(10 Trials) of **0.955**, std 0.009 with max AUC of 0.966. With total parameters of only 51,430. Densenet Siamese has highest mean AUC(10 Trials) of **0.921**, std 0.016, Max AUC, 0.95 With total parameters of only 16,725,485. This total parameter size is so large because of the multiplication of the flattened feature map from each Densenet branch with fully connected layer of Siamese branch and following concatenation of the feature maps from both Densenet branches. Contrastive loss with VGG-Siamese network have results of mean AUC (Ten trials) of **0.944** with std of 0.007 and highest AUC value in a single run as 0.951. With total parameter size of 3,281,840. Though another network structure has scored 0.943 mean AUC, std 0.006 and max AUC of 0.951 with total parameters size of 1,068,080. The difference between the two network is the size of the output of the fully connected layer, 2048 is the fully connected network output size for the bigger network, and for the smaller network that is 128. As an effect the total number of parameters are almost one third but the performance is almost same.

References

[1] Wikipedia(Hyperparameter). Hyperparameter machine learning, November 2018. URL https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning).