Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it 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 Contrastive Loss

1.1.1 Search spaces

- Evaluation script – Done (For VGG like network)

    1. Conv filter sizes – To analyze
    2. Kernel size – To analyze
    3. Initialization (He_normal, He_uniform, Glorot_uniform(default), Glorot_normal(Also called Xavier normal) RandomNormal.
    4. Layers (single, two, three) – To analyze
    5. Dense filters (32, 64, 96, 256, 512, 1024, 2048, 4096)
    6. Dropouts (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8) or draw at random and evaluate for 100 cases and chose the best option ? – To analyze
    7. Batch normalization (True or False) – To analyze
    8. Batch Size – To analyze
    9. Optimizer (Adam, Nadam, Adadelta, Adamax, RMSprop)
    10. Learning rate (0.01, 0.007, 0.005, 0.002, 0.0007, 0.0005, 0.0002, 0.0001, 0.00007, 0.00005, 0.00002, 0.00001)

11. Initializer for Con2D and dense layer. It is found out that the random_uniform default settings is not converging at all for conv2D initializer.

1.1.2 Initializer

Keras initializers[2] control/define the way the initial random weights in keras layers are set.

Zeros : This is one of the simplest of the initializers. It generates tensors initialized to 0.
Instantiation: keras.initializers.Zeros()

RandomNormal : This Initializer uses normal distribution to generate the tensors.
Instantiation: keras.initializers.RandomNormal(mean=0.0, stddev=0.05, seed=None)

RandomUniform : This Initializer uses uniform distribution to generate the tensors.
Instantiation: keras.initializers.RandomUniform(minval=-0.05, maxval=0.05, seed=None)

Glorot_normal : Glorot normal initializer is also known as Xavier normal initializer. It generates samples from a truncated normal distribution which is centered at 0 with standard deviation (stddev) = sqrt(2 / (fan_in + fan_out)) where fan_in represents number of input units in the weight tensor and the number of output units in the weight tensor is the fan_out.
Instantiation: keras.initializers.glorot_normal(seed=None)

Glorot_uniform : Glorot uniform initializer, also called Xavier uniform initializer. It draws samples from a uniform distribution within [-limit, limit] where limit is sqrt(6 / (fan_in + fan_out)) where fan_in represents number of input units in the weight tensor and the number of output units in the weight tensor is the fan_out.
Instantiation: keras.initializers.glorot_uniform(seed=None)

He_normal : It draws samples from a truncated normal distribution centered on 0 with standard deviation (stddev) = sqrt(2 / fan_in) where fan_in represents the number of input units in the weight tensor.
Instantiation: keras.initializers.he_normal(seed=None)

He_uniform : He uniform variance scaling initializer draws samples from a uniform distribution within [-limit, limit]. Here, limit is sqrt(6 / fan_in) and fan_in represents the number of input units to the weight tensor.
Instantiation: keras.initializers.he_uniform(seed=None)

- The initializers evaluated here are in two different sets. Kernel initializers for the convolution networks and kernel initializers for the Dense layers.
- The kernel initializers evaluated for the convolution layers (conv2D) are He normal and uniform, Glorot normal and uniform and RandomNormal.
- Interesting observation is that with Randomuniform initializer the conv2D does not train at all.
- For the Dense layers He normal and uniform and Glorot normal and uniform is used
- The bias initializer, for both con2D and Dense layers, is used with default 'Zeros' option.
- In keras for both conv2D and Dense the default kernel initializer is Glorot uniform
- Popular Intuition is that Glorot or Xavier initialization works better with Sigmoid, while He uniform/normal works better with Relu.
- In our case there are 13 Conv2D layers compared to only one or two dense layers, so the conv2D initializer is expected to have more effect.
- All the initializers used with seed None. Because we did not know which seed value to provide for the best result.
- Initializer for Con2D and dense layer. It is found out that the random_uniform default settings is not converging at all for conv2D initializer.

In this graph, observed mean AUC is reported for different combination of conv2D initializer and Dense Initializer. The x axis of the graph represents dense layer initializer and the y axis represents the conv2D initializer. In the z axis the

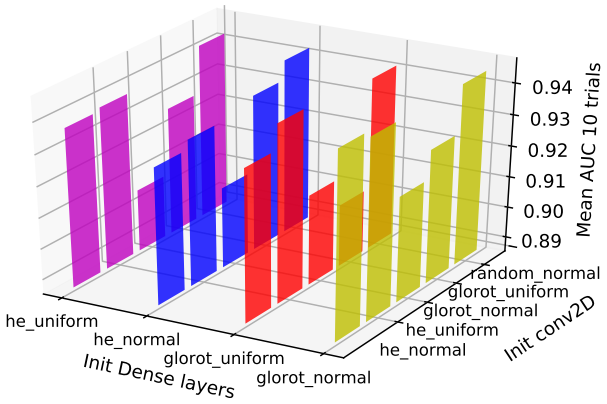

Figure 1.1: Comparison of initialization of conv2D and Dense layers

mean AUC of ten trials is shown. Please note that the z-axis values are clipped and starts from 0.89. This is to be able to show the bar heights differences effectively. Otherwise, if the bars were plotted from zeros they all look similar height as the values are very close indeed. Lowest mean AUC value obtained is 0.903 and the highest is 0.943. The bar chart starting value selected in such a way that all the bars are clearly visible and comparable. Now the Interesting observations from the graph are as follows.

- The performance of Randomnormal as the initializer for conv2D layers over all good.
- Performance of the Glorot normal and uniform both as conv2D initializer is comparitively bad than others.
- Performance of He is over all quite good as a conv2D initializer.
- As the Dense layer initializer glorot normal is found to have performed the best.
- Over all performance wise the He normal as the conv2D and glorot normal as the dense Initializer has performed slightly better 0.943 than second best

Rank	Init conv2D	Init Dense	Mean AUC	STD	MAX AUC
1	He normal	Glorot normal	0.943	0.007	0.953
2	He uniform	Glorot normal	0.941	0.012	0.961
3	RandomNormal	Glorot normal	0.941	0.004	0.946
4	He uniform	Glorot uniform	0.94	0.008	0.952
5	RandomNormal	He normal	0.939	0.005	0.946
6	RandomNormal	He uniform	0.939	0.005	0.948

Table 1.1: Kernel initializer top results

0.941 with (He uniform, Glorot normal), and also (RandomNormal, Glorot normal) both. The best mean AUC results are shown below.

- From table 1.1 it is observed that there might be a trend that with RandomNormal kernel initializers, the standard deviation is very low, but the max AUC values are consitantly lower than the He counter parts. However all the top ranking results are very close in terms of mean AUC.

1.1.3 Learning rate and optimizer

During the training, in backpropagation step the anlytic gradient is computed which is used to update the parameters of the network. This update stage could be done in different ways, this is where the optimizer come into action. While the main target of the deep learning task is to find the minima, the optimizers can control how soon or robustly the minima is found. There is a very compelling comparison of optimization process to a ball or particle rolling down hill in the Stanford lecture series [1]. It compares the loss function to a hill and randomly initializing the weights to particle with zero velocity at random points on the hill. Now the optimization process is compared to simulating the particle's motion (parameter vector) of rolling down the hill landscape (loss). Keras sources [3] gives very brief description of the optimizers. Important optimizers are as follows.

SGD Stochastic gradient descent optimizer. The very first of it's kind, conceptualized by H. Robbins and S. Munro back in 1951. Even though it remains one of the

most preferred optimizer till date (different variations i.e with momentum, Nesterov etc), this optimizer is not evaluated in this work in favor of more theoritically advanced optmizers.

Adagrad Instead of globally varying the learning rate, the concept of per parameter adaptive learning rate was first introduced by Duchi et al. in Adagrad optimizer. It seems it has a limitation though, the use of monotonic learning rate is often too aggressive and the learning stops too early. This optimizer is also not included in this study in favor of more adavanced optimizers.

RMSprop RMSprop try to compensate the aggressive monotonically decreasing learning rate from Adagrad by introducing the moving average of squared gradient.

Adam Adam can be seen as RMSprop with momentum.

Nadam It incorporates Nesterov momentum into Adam.

Adamax Adamax is a variant of Adam which uses infinity norm.

Adadelta It is like Adagrad with moving window of gradient updates.

Results

It is very Interesting to note that all the optimizers were found to be working very good and the best mean AUC result for each of the optimizers are rather close. But for different learning rates. Nadam is the best performer here. Not only the best mean AUC is highest but also the max AUC of one of the ten trials is the highest. So we are selecting Nadam for finer evaluations. Since all the results are close does not make lot sense to evaluate the best network for all the optimizers.

The top results are as follows: (Adam+0.00001, Nadam+0.0002, Adadelta+0.005, Adamax+0.0007, RMSprop+0.0002)

Please note, this graphs are clearer than before because now it's been done in matplotlib, included as PDF.

1.1.4 Final gridsearch

1. Conv filter sizes – 16-16-32-32
2. Kernel size – 3

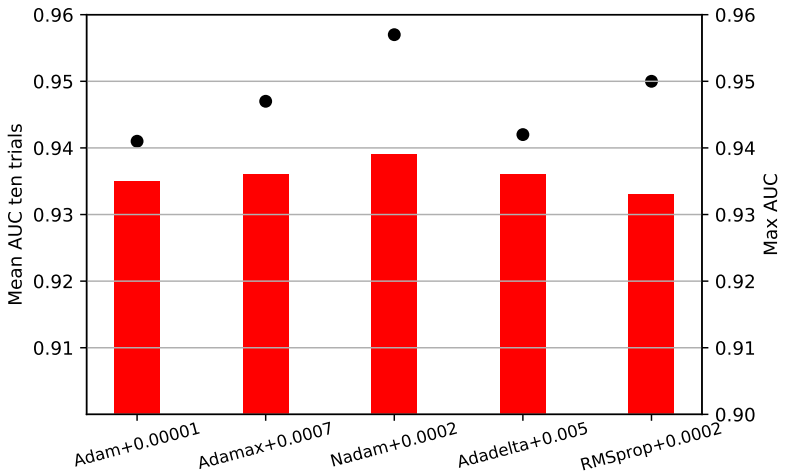

Figure 1.2: Best optimizer and learning rate

3. Initializers – No clear winner (he normal+glorot normal), (he uniform+glorot normal), (RandomNormal+glorot normal)

4. Layers (single, two, three) – single, two is very close most of the time though. But single layer is lesser parameters, right? TODO: total parameters analysis

5. Dense filters 128, 2048. In between results are close, before 128 results are worse. 2048 gives very less standard of deviation but may be the max value is too low. Evaluating both.

6. Dropouts 0.4

7. Batch normalization (True or False) False

8. Batch Size – 256

9. Learning rate and optimizer Nadam and 0.0002

1.1.5 Current status

- Final Grid search is running. It has been observed that the training accuracy has been very low, between 70 to 80% only. But the test data mean AUC score is very high as 0.93+

- What is convergence criteria? If we are training the network longer and it can achieve greater fit or accuracy to train data but the test AUC score goes down. How to determine where to stop. Is it fair to evaluate the network for different epochs and where it gives best mean test accuracy select that fixed epoch for the evaluation?

1.1.6 Next

- 20 January is the submission day. Deadline first/final draft? Is there review? – 10 January draft ready for peer review
- Monte Carlo dropout analysis for all the networks. (I need to read more on this)
- Sanity checks on the already tested networks, such as reorder the image pairs and see how it affects the training. – Prof said, not required
- Prediction analysis on the images. – very important
- Run time/ computation power memory analysis if possible – Good to have
- Do we try another method like Resnet ? – No new method

References

[1] CNN for visual recognition. Cs231n Stanford University, November 2018. URL http://cs231n.github.io/neural-networks-3/#update.

[2] Keras initializers. Initializers keras, November 2018. URL https://keras.io/initializers/.

[3] Keras optimizers. optimizers keras, November 2018. URL https://keras.io/optimizers/.