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-Toro

January 2019

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work.

Date	Arka Mallick

Abstract

Accurately modelling features manually for sonar image matching has always been very challenging. Only recently in Valdenegro et al. [36], instead of hand designed features, a Convolutional Neural Network (CNN) was encoded to learn the general similarity function to be able to predict a pair of sonar patches belong to the different instance of same object or a different one. The result has been a significant improvement over the state of the art methods. However the results need to be further improved. The error in its prediction affects the overall performance of object recognition, tracking etc high level tasks which are based on patch matching. In this work more advanced CNNs are evaluated with the goal of improving the state of the art results. Using DenseNet it was possible to predict binary classification score of matching and non-matching cases with 0.955 average AUC of ten trials, where the network was trained from scratch. This was a clear improvement over the state of the art [36] results of AUC 0.894 on the same unseen test dataset. For Siamese network with DenseNet branches the best performance was 0.921 average AUC (10 trials). A Siamese network with VGG branches were also evaluated along with Contrastive loss. This network's best performance was 0.944 average AUC (10 trials). To ensure the evaluation is effective thorough hyperparameter search has been performed for all three methods.

Dedicated to the memory of my maternal uncle,
Subhas Chandra Ghosh,
without whose support my dream of pursuing higher
studies would not have been possible.
May he rest in peace.

Acknowledgements

My deepest thanks goes to my supervisor Dr. Matias Valdenegro-Toro. For several reasons. From technical aspect, his guidance and suggestions were invaluable and always spot on. At the beginning I spent several months training for deep learning and exploring other prerequisites for this work. He has been very supportive and motivating throughout the period of this thesis and even much before. In spite of very busy schedule, he spent a lot of hours helping me, many times even during weekends. It was a great experience working with him. Also, it was a long time aspiration to work in the domain of underwater robotics. A special thanks to him for giving me this opportunity.

Special thanks Prof. Dr. Paul G. Plöger for making this project possible through his supervision and invaluable inputs, also for the neural networks lesson, which was pivotal for deep learning training. I sincerely thank to Prof. Dr. Gerhard K. Kraetzschmar for kindly agreeing to supervise this project. Along with all the technical lessons, I also thank him for the wonderful opportunity of being able to take part in Robocup competitions, which were great experience for me to learn hands on programming and be in touch with great minds. In the context, my sincere thanks Santosh, Oscar and Shehzad who were great mentors and taught me many important lessons. I also thank Prof. Dr. Rudolf Berrendorf and the entire cluster team for providing such great computational environment for all the students. Without cluster I could have never finished this work.

I must thank my best friends Swarit, Pranika, Meemansa, Arshia, Aaqib and Nour for making my stay in Bonn stress-free and enjoyable. I also thank Adhideb and Abhilash, without their help I would not dare dreaming about masters in Germany. Finally, I thank my family and my partner Arshia for being supportive and encouraging through out and just being there whenever I need them.

Contents

List of Figures

List of Tables

1

Introduction

"You can't cross the sea merely by standing and staring at the water."

- Rabindranath Tagore

More than two-thirds of the earth surface is covered by ocean and other water bodies. For a human it is often impossible to be able to explore it extensively. For finding new source of energy, monitoring tsunami, global warming or may be just to learn about deep-sea eco-system or may be to look for a lost ship or an airplane, the need for venturing into potentially dangerous underwater scenarios appear regularly, in fact getting more frequent. That's why more and more robots are being deployed in underwater scenarios and lot of research is going in this direction. Some of the exploration or monitoring tasks requires the robot to see underwater, to make intelligent decisions. But underwater environment is very difficult for optical cameras, as light is attenuated and absorbed by the particles in the water. And lot of real life monitoring and mapping tasks take place in cluttered and turbid underwater scenario. The limited visibility range of optical sensor is a big challenge. Hence sonar is more practical choice for underwater sensing as the sound can travel long distances with comparatively little attenuation. Though there are challenges with acoustic sensing as well, for example, it has low signal-to-noise ratio, lower resolution and unwanted reflections also cause problems.

Patch matching is one of the fundamental tasks in todays myriad computer vision and image processing applications. Patch matching can be used as low-level tasks like image stitching [2], deriving structure from motion [28] or high-level task as object instance recognition [25], object classification [41], multi-view reconstruction [33], image-retrieval etc. Patch matching for acoustic images are getting increasingly useful in underwater scenarios for data association in simultaneous localization and mapping (SLAM), object tracking, sonar image mosaicing [16] etc. applications. Typical challenges in patch matching tasks are different viewing points, variations in illumination of the scene, occlusion and different sensor settings. For patch matching in sonar images the typical issues with sonar adds to the overall challenge, such as low signal to noise ratio, less visibility etc., causes the underlying object features to be not so prominent as a normal image. So it has been found that it is very challenging to manually design features for the matching task [36].

1.1 Motivation

In Valdenegro et al. [36] it was first displayed that deep learning methods can be directly applied to the sonar image patch classification task. Without using any hand designed features it is still possible to perform a patch matching task with high accuracy. In [36] the authors recorded Forward-Looking Sonar (FLS) images of different objects in underwater environment and generated balanced dataset of image patches for the classification task. Two channel convolutional network and Siamese convolutional network were used in predicting the binary classification scores (table 1.1) on unseen test data. Using Two channel CNN the overall best result obtained was of Receiver operating characteristic (ROC) area under curve (AUC) of 0.894, with test accuracy of 82.9%. The reported results are better than both classic keypoint matching methods (AUC 0.61 to 0.68) and machine learning based methods such as Support Vector Machine (SVM) and Random Forests (AUC 0.65 to 0.80). These findings were pivotal for the work presented in this thesis.

Network Type	Output	Test Objects	AUC	Mean Accuracy
2-Chan CNN	Score	Different	0.894	82.9%
Siamese CNN	Score	Different	0.826	77.0%

Table 1.1: Best result from Valdenegro et. al. 'Different' represents the underlying test objects were different from the train objects. The binary prediction scores are presented here from the original results in [36]

1.2 Problem Statement

However the results in [36] is far from perfect yet and the mis-classifications affect the object detection, recognition and tracking, which are dependent on the patch matching task. It is desired that the result is further improved. With this goal the same dataset has been obtained from the [36] for further evaluation. In this thesis more advanced architectures (such as DenseNet) [15] will be evaluated on the data and the goal is to improve the state of the art on the dataset.

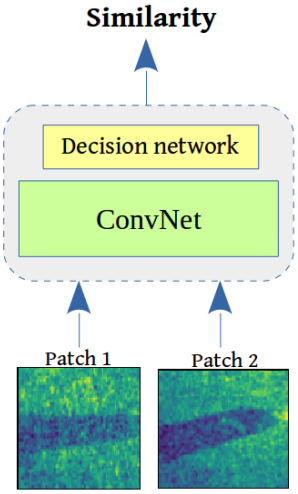


Figure 1.1: Use of convolutional network for learning general similarity function for image patches. The patches in the image are samples taken from the data used in this thesis. Inspired from Zagoruyko et al.[42]

For performing the patch matching task without using any hand designed features such as SIFT [26], we need to encode a network to learn the general similarity function (figure 1.1) for sonar image patches. Which is able to predict that two input sonar patches contain different views of a same object or not i.e matching or non-matching respectively.

With the goal of obtaining the best possible result in the evaluation, best hyperparameter search needs to be performed for each of the networks. The hyperparameters are the parameters of network or learning process whose values are already set before the training starts [40]. For example the starting learning rate or the optimizer can be fixed before the training process start.

Performing best hyper parameter search for each of the architectures is very important part of the evaluation. Another objective of this work is to do a comparative analysis of the performance of all the architectures.

2

State of the Art

2.1 Convolutional Neural Network

Before starting with the state of the art, little introduction to the Convolutional neural network (CNN) is provided here. CNN [39] is a special type of Artificial Neural Network with Multi-Layer Perceptrons (MLP), consisting of learnable weights and biases. CNNs are usually trained with back propagation algorithm in a supervised manner. While a end-to-end CNN also expresses a single differentiable score function similar to a normal ANN, CNNs are different in architecture though, specially designed for recognizing visual patterns directly from the raw image pixels with minimal preprocessing [23]. Unlike fully-connected architecture, in CNN, neurons in a layer will only be connected to a small region of the layer before. CNNs take advantage of the explicit assumption that inputs are images, as a result they can be more efficiently encoded to have much lesser parameters. The main building blocks of CNNs are Convolutional Layer (Conv), Pooling Layer, Fully-Connected Layer (FC). Now a typical example of a CNN can be denoted by `[Input - Conv - ReLU - MP - FC]`, where `Input` holds raw pixel values of the input image, `Conv` layer applies convolution operation to the local region of input and produce output; this operation is inspired by the natural response of a neuron in visual cortex getting stimulated. `ReLU` layer provides activation to each elements and applies a threshold (`max(0,x)`) so that the negative weights become zero. `Pooling` layer which basically down-samples the input along the

spatial dimensions of supplied height and width, by mapping a cluster of neurons from one layer to a single output in the next layer. `MP` or max-Pooling layer takes the maximum value from the cluster of the neurons and maps in the next layer. `Average` pooling takes average values of the cluster and maps to the next layer. Pooling can also be local or global. Lastly the `FC` layer just connects every neurons of one layer to the next layer, just like MLP. Fully-Connected layers serve as the decision network and determines the overall output size.

Inspired by [36] following notation for CNN layers are used to describe components of the architectures: `Conv(Nf,Fw x Fh)` is a convolutional layers with Nf filters of width Fw and height Fh. A max-pooling layer is represented by `MP(Pw, Ph)` where `Pw x Ph` is the sub-sampling size of the layer, and FC(n) is a fully connected layers where n represents output size.

In the following section Siamese network is briefly discussed because it is important to have a understanding of it before some of the state of the art network architectures can be explained in details.

2.2 Siamese Network

In 1993 the concept of a new artificial neural network called Siamese network (the network in right in figure 2.2) was introduced by Bromley et al. [1]. Siamese network consists two identical sub-networks which are joined at the output. During training stage one input each is connected to each of the subnetworks. The main idea here is that the subnetworks (also called branches) are trained simultaneously and extract features from the inputs. While the shared neurons are capable of measuring the distance between the two extracted feature vectors by each branch. If the predicted distance between two feature vectors is lesser than a threshold then it can be considered that the inputs are similar. Authors used this concept to compare two signatures in [1]. One of the signature was previously obtained from the authentic owner (up to 6 signatures were recorded). This was then used to compare with a new signature to verify if the both persons are same or not, as precaution against forgery. Authors implemented the Siamese network to be able to extract and compare different features of the two signatures and if the output is within a threshold then they were considered matching. If not then it was most

likely a forgery.

2.3 Sonar image matching techniques

Sonar image patch matching is more difficult than normal optical patch matching problem. This is because sonar images have additional challenges such as, non-uniform insonification, low signal-to-noise ratio, poor contrast, low resolution, low feature repeatability [17] etc. But sonar image matching has important applications like in sonar registration, mosaicing [21], [16] and mapping of seabed surface [29] etc. While Kim et al. [21] used Harris corner detection and matched keypoints to register sonar images, Hurtos et al. [16] incorporated Fourier-based features for registration of FLS images. S. Negahdaripour et al. [29] estimated mathematical models from the dynamics of object movements and it's shadows. Vandrish et al. [37] used SIFT [26] for sidescan sonar image registration. Even though these approaches did achieve considerable success in respective goals, those were found to be most effective when the rotation/translation between the frames of sonar images are comparatively smaller. Block-matching was performed on segmented sonar images by Pham et al. [30], Self-Organizing Map was used for the registration and mosaicing task using sidescan sonar images. Recently, [43] for stereo matching, [20] for fast under-water object detection and localization, [35] objectness scoring, CNNs are being more and more used for sonar image processing. The main reason behind such rise of CNN usage is that, the CNNs can learn sonar-specific information from the sonar data directly. No complex manual feature design or rigorous data preprocessing steps are needed, which makes the task less complex but prediction accuracy achieved is higher.

2.4 Learning similarity function

Zagoruyko et al. have demonstrated that CNNs can be directly deployed to learn the underlying similarity function to be able to determine that two input images/patches are different instances of same object or not, without any help from hand designed features. The scarcity of accurate hand designed features for sonar images motivated Valdenegro et al. to evaluate similar approach as Zagoruyko et al. and encode a CNN for sonar image comparison.

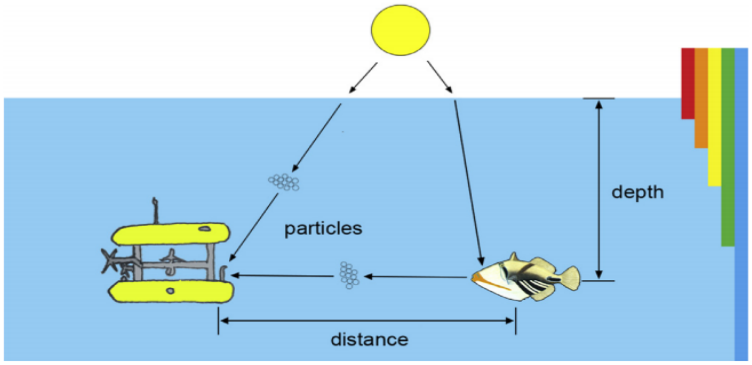


Figure 2.1: Figure is taken from S. Emberton et al. [11]. Light gets absorbed and scattered by the particles and objects in the water, which causes poor contrast and spectral distortion in sonar images.

Valdenegro et al. [36] evaluated two architectures, a two-channel network and a Siamese network. The architectures 2.2 were based on the work from Zagoruyko et al. [42]. A grid search was used over pre-defined set of parameters which define the network structure. The final two channel network structure presented in the paper for predicting binary classification score was as follows, `Conv(16, 5 x 5)-MP(2, 2)-Conv(32, 5 x 5)-MP(2, 2)-Conv(32, 5 x 5)-MP(2, 2)- Conv(16, 5 x 5)-MP(2, 2)-FC(64)-FC(32)-FC(1)`. Similarly grid search was also conducted for Siamese network structure for classification score. The structure for each of the branches or sub-network is as follows, `Conv(16, 5 x 5)-MP(2, 2)-Conv(32, 5 x 5)- MP(2, 2)-Conv(64, 5 x 5)-MP(2, 2)-Conv(32, 5 x 5)-MP(2, 2) -FC(96)-FC(96)`. The output features from the branches were then concatenated to form 192 element vector. This was then passed through a decision network with `FC(64)-FC(1)`. For both two channel and Siamese network the final `FC(1)` layer had Sigmoid [8] activation function and binary cross entropy loss [5] function.

Not very complex network structures were used. In general sense, adding more layers in the network does add more non-linearity to it and in some cases, that improves the performance of the network. So in theory, more advanced networks or loss functions which help extracting more discriminative and patch invariant features, should improve the results even further.

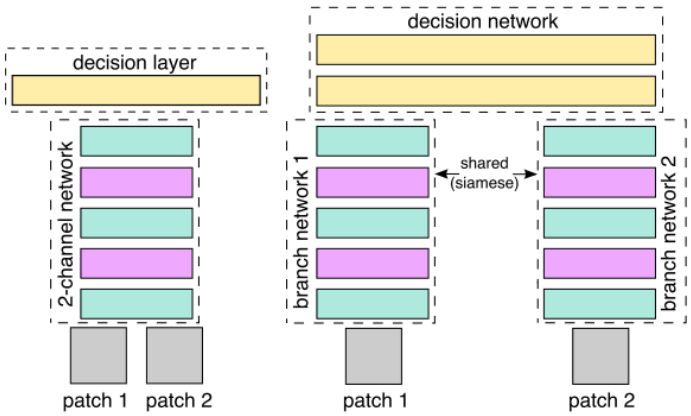


Figure 2.2: Two channel network (left) and Siamese (right) CNN architectures used in Valdenegro et al., though the figure and inspiration is from [42]

3

Methodology

3.1 Data

Just like any other machine learning project the data is the most important part of the evaluation. Hence for the proper qualitative evaluation a deeper look at the data and how it was generated is needed. The data set for training and testing are both obtained from Valdenegro et al [36]. Following is a brief description of the overall pipeline to obtain the data for training.

3.1.1 Sonar image capture procedure

In Valdenegro et al [36] the authors have explained in details how the raw sonar images were captured. In a water tank in their facility, the images were taken using ARIS Explorer 3000 and mounted Forward-Looking Sonar (FLS). The objects featured are household garbage items and common marine debris. There are total 9 different types, such as metal cans, bottles, drink carton, metal chain, propeller, tire, hook, valve. In [36], in controlled underwater environment total of 2072 images; about 2500 instances of the aforementioned objects were captured and labeled with 9 classes accordingly.

$L_m \leftarrow \emptyset, L_{nm} \leftarrow \emptyset$
for img $\in I$ **do**
  for object $o \in$ img **do**
    $OC \leftarrow$ crop B_o from img.
    for $i = 0$ to S_p **do**
      $MC \leftarrow$ sample random object p of class C_o and make an image crop.
      Append (OC, MC) to L_m
    end for
    for $i = 0$ to S_n **do**
      $NMC \leftarrow$ sample random object p of class $C_p \neq C_o$, and make an image crop.
      Append (OC, NMC) to L_{nm}
    end for
    for $i = 0$ to S_n **do**
      $BC \leftarrow$ sample random background patch and make an image crop.
      Append (OC, BC) to L_{nm}
    end for
  end for
end for

Figure 3.1: Algorithm for matching and non matching pair of patch generation from Valdenegro et al [36]

3.1.2 Data generation for deep learning

As part of the same work [36], matching and non-matching pairs of sonar image patches were generated using a patch generation algorithm displayed in figure 3.1 where each patch, an instance of one of the 9 classes, were obtained from meaningful crops of the original sonar images. Authors have figured that 96x96 pixels is the best size of the crops.

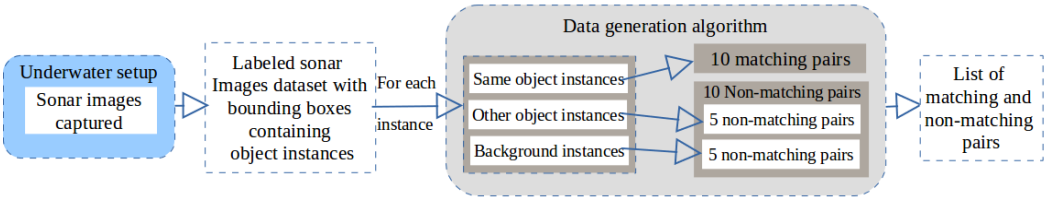


Figure 3.2: Data processing pipe line

To create matching pair, two patches that belong to the same object-type but different perspective and insonification level were used. For generating non-

matching pair, two instances of different object-types were used. Also, one instance of an object and a background patch, which contains no object instance, were used to generate additional non-matching patches. According to the authors of [36], for balanced and effective learning, the ratio of matching and non-matching pairs in both train and test data is maintained at 1:1. That is, for every ten matching pair five object-object non-matching and five object-background non-matching pairs were generated. In figure 3.2 the overall pipeline for the data generation is displayed in simple block diagram. In figure 3.3 some sample patches from all type of pairs are displayed. Using the patch generation algorithm total of 39840 pairs were generated from the instances of 6 distinct object type, which were used as the train data. While another 7440 pairs were generated from the instances of remaining object types, for testing purpose. This test dataset does not contain any common object with the training dataset, it should be a good test for the generalization of the approaches. The labels for the data are 0 and 1 representing non-matching and matching pairs respectively. For this thesis work, this dataset is obtained in HDF5 files [14]. It contains patches in form of tensors [34], each of shape (2,96,96). Here the pair of patches (size 96x96) are placed in a way that each channel contains one patch.

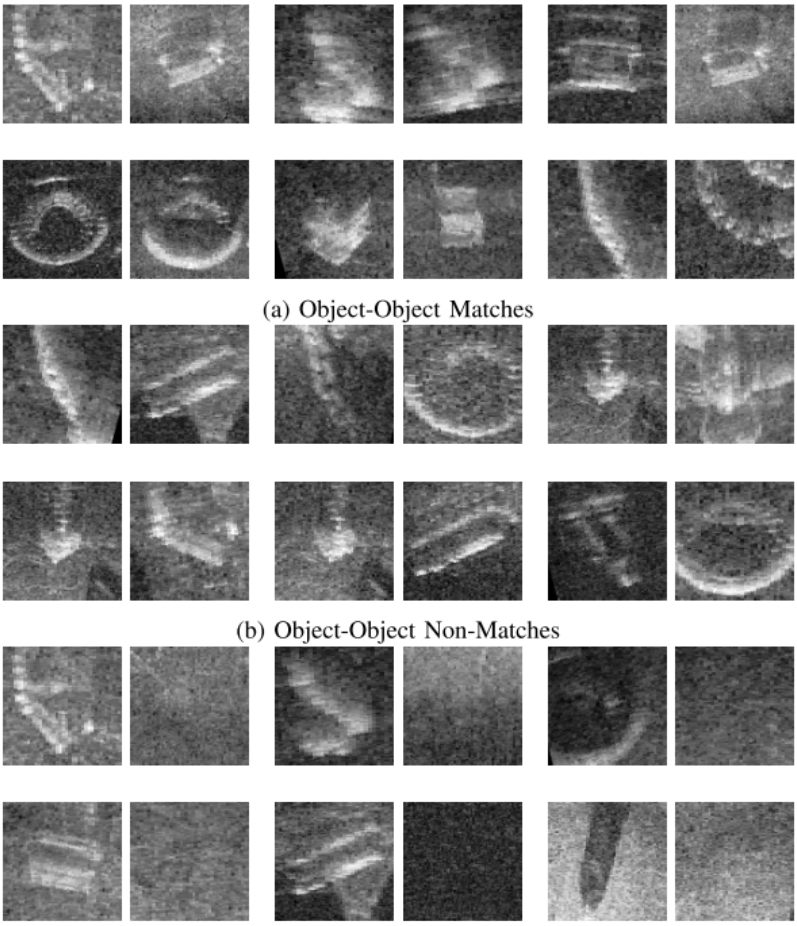


Figure 3.3: Some sample of the matching and non-matching pairs from Valdenegro et al [36]

3.2 Architectures

In this section the theoretical aspect of the different architectures that we are using are explained along with brief implementation details where applicable.

3.2.1 Densenet two channel network

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.

The Densenet implementation used is from the keras community contributions (keras_contrib) [27]. Another Densenet implementation was initially evaluated from Thibault de Boissiere [9], but it was older implementation and did not have bottleneck or compression implemented.

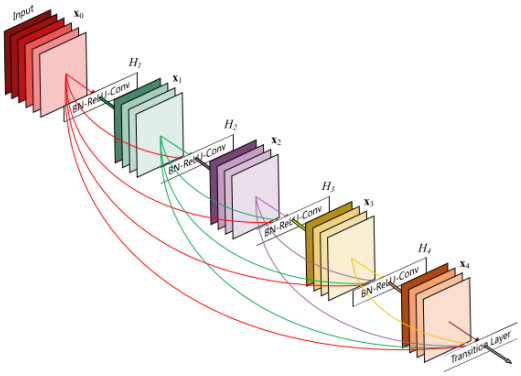


Figure 3.4: Example DenseNet architecture from [15].

In Densenet two channel the the sonar patches are supplied as inputs in two channels format, the network by itself divides each channel into one patch and learn the features from the patches and then finally compare them using the Sigmoid activation function at the end with FC layer of single output. The original label in the data is 0 for non-matching pair of input patches. Label 1 means matching pair.

Densenet architecture depends on the hyperparameter settings of the instantiation. The parameters and their brief definition is described in the following section.

- **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.

3.2.2 Densenet Siamese network

In this architecture the branches of the Siamese network are Densenet. Following the classic Siamese model the branches shares weights between them and gets trained simultaneously on two input patches and then learns the features from the inputs. Through the shared neurons it the Siamese network is able to learn the similarity function and be able to discriminate between the two input patches. The role of the Densenet branches are feature extraction, the decision making or prediction part is taken care of by the Siamese network.

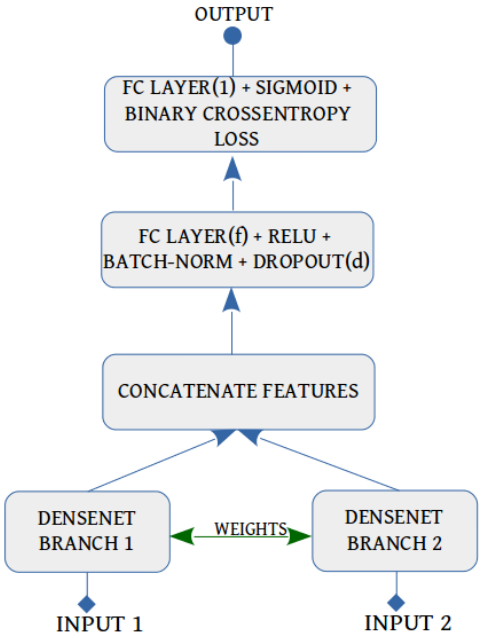


Figure 3.5: Siamese Densenet structure.

In figure 3.5 the basic architecture is displayed for the Densenet Siamese network. The two Densenet branches are designed to share weights between them. The extracted features are concatenated and connected through a FC layer, followed by activation Relu and where applicable Batch normalization and dropout layers. The output is then connected to another FC layer with single output, for binary prediction score of matching (1) or non-matching (0). Sigmoid activation function and binary cross entropy loss function is used for this final FC layer. As mentioned

in the figure 3.5 the size of the output of the FC layer (f) and value of dropout probability etc. hyperparameters are to be decided after thorough hyperparameter search.

3.2.3 Contrastive loss

Using contrastive loss higher dimensional input data (e.g. pair of images) can be mapped in the much lower dimensional output manifold, where similar pairs are placed closer to each other and the dissimilar pairs have bigger distances between them depending on their dissimilarity. So using this loss function the distance between two input patches projected in the output manifold can be predicted and if the distance is closer to 0 then the input pairs are matching, otherwise its dissimilar (above threshold). The formula for the loss is shown below.

$$D_w(\overrightarrow{X_1}, \overrightarrow{X_2}) = \|G_w(\overrightarrow{X_1}) - G_w(\overrightarrow{X_2})\|_2$$

$$L(W, Y, \overrightarrow{X_1}, \overrightarrow{X_2}) = (1 - Y)\frac{1}{2}(D_w)^2 + (Y)\frac{1}{2}\{max(0, m - D_w)\}^2$$

Here L is the loss term, the formula presented here is the most generalized form of the loss function, suitable for batch training. $\vec{X}_1$, $\vec{X}_2$ represents a pair of input image vectors. Y are the labels, 0 for similar pair and 1 for dissimilar pair. D_w is the parameterized distance function to be learned by the algorithm. m is the margin and m is always greater than zero and it defines a radius around G_w. The dissimilar pairs only contribute to the loss function if their distance is within the radius. One of the idea for evaluating this loss function is to use it with a Siamese network, as the loss function takes a pair of images as input. So its very relevant to the problem statement of this thesis.

To evaluate the contrastive loss a Siamese network has been selected since it also takes two input images and compare them. For the branch structure of the Siamese (figure 3.6) VGG network has been chosen. The role of the VGG network is to extract features, similar to the Densenet Siamese, the final decision making and prediction is taken care of by the Siamese network. In this case the output is actually euclidean distance between the two input sonar patches, projected into lower dimension using contrastive loss.

- Explain here about the Siamese network with VGG branches used and why Siamese network is chosen to be evaluated with the contrastive loss
- Do a comparative study with binary cross entropy too and see how that goes, probably it's difficult now, another way is to use this with the Siamese Densenet network and see.

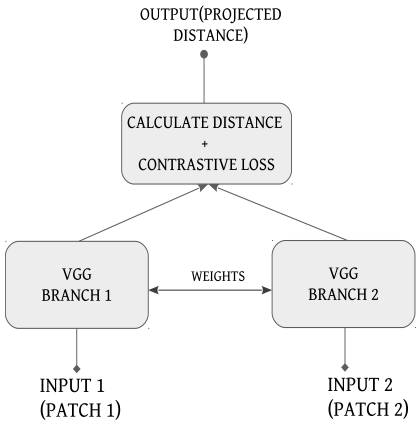


Figure 3.6: VGG Siamese network with contrastive loss

The basic VGG network structure is as follows, Conv(n, a x a)-Conv(n, a x a)-MP(2, 2)-Conv(2n, a x a)-Conv(2n, a x a)-MP(2, 2)-Conv(4n, a x a)-Conv(4n, a x a)-Conv(4n, a x a)-MP(2, 2)-Conv(8n, a x a)-Conv(8n, a x a)-Conv(8n, a x a)-MP(2, 2)- Conv(8n, a x a)-Conv(8n, a x a)-Conv(8n, a x a)-MP(2, 2)-FC(d). The a, n and d are hyperparameters which needs to be optimized. The details of the evaluation is presented in the section 5.3

3.2.4 VGG network

Implementation taken from https://hackernoon.com/learning-keras-by-implementing-vgg16-from-scratch-d036733f2d5

4

Evaluation

4.1 Implementation and measurements.

- Compare based on AUC
- What is roc AUC
- AUC vs Accuracy, why AUC is better than Accuracy or other point based methods.
- Talk about tensors, tensorflow and keras here too
- Criteria of comparison, mainly the mean AUC then max AUC, then number of parameters and execution time*

4.1.1 Search for best hyperparameters

What are the hyperparameters. Effect of setting best hyper parameters.

4.1.2 Grid search

Currently the search space for the evaluation is hand designed and supplied to the evaluation script externally. Example

Figure 4.1: Hyper parameters word cloud.

Epochs	Optimizer	Batch size	Dropout	Use batch norm
5	adam	64	0.4	True
5	adam	64	0.4	False

Table 4.1: Custom grid search space example.

In this example it is displayed that how the search space is hand designed to evaluate different cases, here the effect of using batch normalization can be investigated by comparing the results of two test cases. All the values set in the grid passed as input to the actual code and defines the structure or different parameters from inside the code. In 4.1 the flag use batch norm is set to 'True', which gets propagated in the actual block of code where it enables the use of batch normalization layer, where applicable.

For the evaluation the epochs are set after multiple testings in order to ensure, to some extend, the networks does not get too overtrained and the generalization gets poorer. Also if the network is undertrained, then it will not generalize well. The setting the epochs were one of the big challenge of this work, because in some cases the networks gets overtrained after 4-5 epochs.

4.1.3 Training process

The train data has been divided into train and validation data randomly using sklearn train test data split [32] in 8:2 ratio. This is done using fixed seed to

ensure the validation and train cases are same for all the evaluations across all three methods. It is important to clarify that the roles of the validation data in keras is different than classic training method. In keras the validation data is not used to update the weights. Instead this validation dataset can be used to monitor the network performance in terms of validation accuracy and validation loss. Using callback functions such as `EarlyStopping` and `ReduceLROnPlateu` [3] the network validation performance can be monitored after each epochs of training. By configuring the callbacks properly (parameter: Early stop patience) the training can be stopped if the validation accuracy or loss does not improve after a number of epochs then the training process terminates. Similarly, if the validation performance does not improve after some epochs the learning rate can be reduced by a previously determined factor using `ReduceLROnPlateu` to come out of the local minima. The patience values are set after some manual trials.

```
#Early stopping
es = EarlyStopping(monitor='val_acc', patience=es_patience,verbose=1)
#Model check point
checkpointer = ModelCheckpoint(filepath=weight_file, verbose=2,
    save_best_only=True)
#Learning rate reducer on plateu
lr_reducer = ReduceLROnPlateau(monitor='val_loss', factor=np.sqrt(0.1),
    cooldown=0, patience=lr_patience, min_lr=0.5e-6,verbose=1)
```

5

Results

In this section the evaluation results of all three architectures and related best hyperparameter search results and detailed analysis is presented. Also the architectures are compared to each other based on the prediction on test data.

5.1 DenseNet Siamese network

How the search for best hyperparameters for the DenseNet Siamese were designed and corresponding results are presented in the following section.

5.1.1 Hyperparameters to be evaluated

The whole search space is divided into smaller blocks such as hyperparameters for DenseNet, hyperparameters for Siamese and others.

Hyperparameters of DenseNet

Layers per block defines both the depth of the DenseNet, which is automatically calculated and also defines the number of dense blocks. Besides growth rate, reduction, bottleneck, number of filters (nb_filter), pooling, include top, dropout, subsample initial block and weights are the most of the hyperparameters needed to be defined to instantiate a DenseNet. While some of this values will be fixed for all the evaluation, some needs to be chosen from a possible set of value, which could be infinite as well.

Hyperparameters of Siamese Network

Rest of the Siamese network that serve as decision network that connects the two DenseNet branches have the hyperparameters as follows. FC layer output size, dropout probability value.

Other hyperparameters

Apart from the aforementioned ones there are other general hyperparameters such as learning rate, batch size for training and optimizer. In the following part of the report it is described that how the whole search space is divided into smaller, well defined parts. The results are visualized with help of charts and tables to help in decision making.

5.1.2 DenseNet growth rate and layers per block analysis

Most important part of this network is the feature extraction capability of the DenseNet branches. The overall performance of the network depends on it. So the first focus of the hyperparameter search is to find out the main parameters defining the DenseNet architecture. The overall search space for this could be very big or even infinite. So for the first evaluation we define a coarse search space. With hope to find a best performing parameter configuration or at least narrow down the search space. Layers per block are chosen among 2, 3, 4, 5, 6. For single dense block evaluation goes up-to 12 layers. More than that(14) causes memory to run out as the size gets too big for the cluster gpu memory(16GB). Each network has been evaluated for growth rates of 6, 12, 18, 24, 30, 36. Different dense block sizes of 1, 2, 3, 4. The parameters compression/reduction and bottleneck are set 0.5 and 'True' respectively. Both this parameters control the compactness of the model and help reducing the parameter required, hence in theory, making it possible to evaluate much bigger networks without running into memory shortage issue. The network that is being evaluated here are named DenseNet-BC by authors, i.e DenseNet with bottleneck and compression.

There are other parameters but number of dense block, growth rate and layers per block and reduction ratio are the main parameters which controls the architec-

ture and parameter size of the network the most. The goal of this focused search is to narrow down the overall search space from the DenseNet parameters perspective. For more fine grained analysis the compression and bottleneck parameters will also be evaluated.

DenseNet parameters number of filter (nb_filter) value are fixed at 16 for this search. The parameter classes are set to 2, which represents 1 for matching and 0 for not-matching pairs. 96, 96 is the input image dimensions. And it is single channel. So depending on local setting of the keras, 'channel-first' or 'channel-last' suitable input shape is chosen automatically as (1, 96, 96) or (96, 96, 1) respectively.

Subsample initial block enables the sub sampling of the initial input image to reduce the computation cost. The value is set to 'True'. The DenseNet parameter 'weights' value is set to 'None' to ensure that previously trained weights are not used. The decision network is not required for the branches as Siamese provides that, the value for parameter include top is set to 'False'. The learning rate used for the test was 2E-4. As a regularization measurement dropout probability value for DenseNet used as 0.2 to handle over-fitting. To ensure the grid search is effective, too much regularization is not good, as it can be restricting the overall evaluation at times. Epochs can be different for each architectures to ensure that the networks are able to achieve good training accuracy. But networks should not be over training, so the choice of epoch was selected after multiple manual trials for each set of network configuration in the search space. From the Siamese side of the parameters, after concatenating the DenseNet branch output feature maps, the combined features then passed through a fully-connected layer of 512 output size, which is followed by 'ReLU' activation and batch normalization (BN) and then a dropout layer with probability 0.5 has been added to ensure better generalization. Some of this values could have been further evaluated, how ever the values were obtained after lot of manual tuning and assured to be a decent starting point. 'Flatten' is used as pooling at the end of the DenseNet branches. Instead of the global average pooling from the original implementation. This causes increase in parameters overall though. Because with 'flatten' the multidimensional feature map at the end of DenseNet branch is just flattened. Where as in global average pooling [24], apart from the channel other dimensions are simply collapsed. But it was found that with 'flatten' the network is able to achieve much higher training

accuracy and generalization too for this network. Binary cross entropy 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 trained from scratch.

Growth rate and layers per block search setup summary

The overall search space is summarized in the section below.

Fixed hyperparameters

Other hyperparameters that are needed to instantiate the DenseNet are set to fixed values 5.8 after manual trials, in order to focus on the layers per block and growth rate parameters.

Table 5.1: Fixed hyperparameter values for the evaluation setup.

Name	Value	Name	Value	Name	Value
Number of filter	16	Subsample initial block	'True'	Weights	'None'
Dropout rate	0.2	Include top	'False'	Compression	0.5
Bottleneck	'True'	Pooling	'flatten'	Transition pooling	'max'
Siamese FC output	512	Siamese dropout	0.5	Optimizer	'adam'
Learning rate	2E-4				

Varying hyperparameters

- **Layers per block**:
  - **One dense block architecture (nb_dense_block=1)**
    '2', '3', '4', '6', '8', '10', '12'
  - **Two dense block architecture (nb_dense_block=2)**
    '2-2', '2-3', '2-4', '3-3', '3-4', '3-5', '4-4', '6-6'
  - **Three dense block architecture (nb_dense_block=3)**
    '2-2-2', '2-2-3', '2-3-3', '2-2-4', '2-3-4', '3-3-2', '3-3-3', '3-3-4', '3-4-4', '3-4-5', '3-3-6', '4-4-4', '4-4-2', '4-4-3', '4-4-6', '6-4-2', '6-6-3', '6-6-6'
  - **Four dense block architecture (nb_dense_block=4)**
    '2-2-2-2', '3-3-3-3', '4-4-4-4', '6-6-6-6'

- **Growth rate**:
  - Thin layers: 6, 12, 18
  - Thick layers: 24, 30, 36

The evaluation result is thoroughly analyzed and presented in the following section. Since the search space was big and each network configurations were evaluated 5 times, there are lot of data which are analyzed part by part with specific goals in mind.

Performance comparison based on mean AUC

Each test case is trained from scratch and evaluated on test data 5 times. The metric for evaluation is Area under curve(AUC) for the test data prediction. All together there are too many results to display in report. So only **top 20** configurations with highest mean AUC on test data across 5 trials are selected and displayed.

Discussion

From figure 5.1 it is observed that the DenseNet with two dense blocks or layers (e.g., 2-2, 3-4) works best, ahead of single layer ones. Performance of the three and four layer DenseNets are not good. This is unexpected. According to the original paper [15] the performance of a normal DenseNet increases as more deeper the network gets. However, in the original work the DenseNet is used individually as the feature extraction and decision network both. In this case, DenseNet is only used as the feature extraction network. Authors of [15] though hinted that the depth of the network depends on the data volume available too. For example for ImageNet [10], authors used four layer DenseNet with high growth rates. But they used three layer DenseNet in other cases mostly. In any case when the DenseNet two-channel network is evaluated this issue can be verified with more conviction than with DenseNet Siamese, simply because the DenseNet has been used in different way in this case. From figure 5.1 no strong trend for growth rate was observed, so further analysis or some other view of the data needs to be looked into. The network was evaluated 5 times for each configurations because it was observed that not every

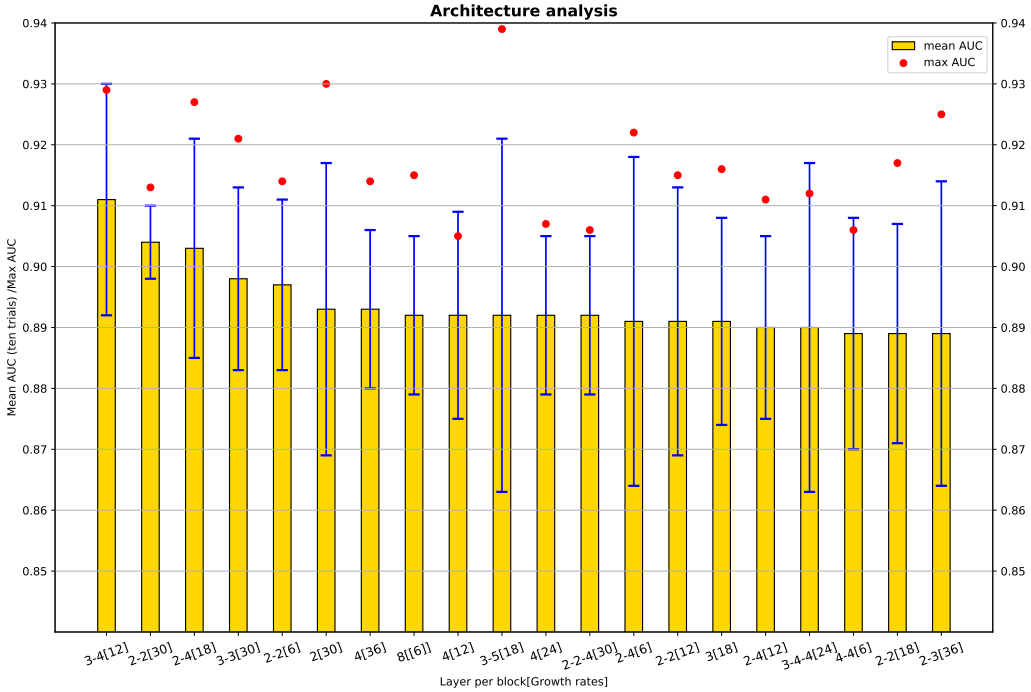


Figure 5.1: DenseNet layers per block and growth rate analysis, sorted by mean AUC, high to low. In x-axis the layers per block and growth rate are displayed together, with growth rate in brackets. It is easy to point out that, the top 20 is dominated by the two layers architectures. The four layer ones do not come close at all. Just few three layer and some single layer architectures are showing good results.

time the network performs exactly same way. Some times it score much higher and some times it might even get stuck in a local minima. The weights of the network are randomly initiated using default initializer Glorot uniform [6]. However many parameters are involved and because every time the weights are drawn randomly and the network gets trained from scratch, the decision boundary at the end of same number of epochs may look very different. Since there are too many networks to be evaluated only 5 times each of them were evaluated, with the idea that the networks with good performance can be evaluated again for higher number of times to verify their consistency.

Performance comparison based on maximum AUC

There were some networks, specially three layer ones, which had comparatively poorer mean AUC but at least one of the 5 runs they had scored very high AUC. That is why all the architectures were sorted according to their maximum AUC in one of the 5 trials. Displaying below in figure 5.2 is the **top 20** architectures (layers per dense block and growth rates) in terms of highest AUC on test data across 5 trials.

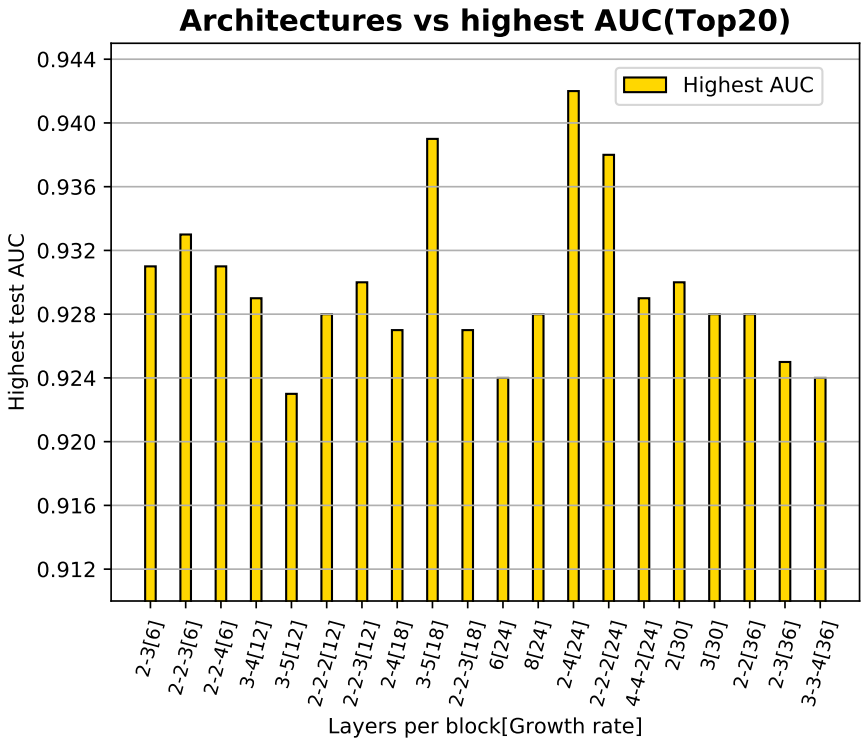


Figure 5.2: DenseNet layers per block and growth rate analysis, top 20 maximum AUC, sorted by growth rate. It seems the architectures growth rate 24 is most frequent here ahead of 12 and 18. unlike the top 20 mean AUC analysis many three layer architectures are in this list.

Discussion

In both top 20 lists mentioned above the four dense block architectures did not make it in any of the case. Their performance on the test data is worse, so

under current setup and assumptions they work worse than lesser block networks. Even though four layer networks are able to train above 95% train accuracy their generalization on the test data seems to be poor in general. Two dense block networks in general works best in terms of mean AUC of the 5 evaluations. In terms of max AUC score some of the Three block and one block DenseNet works very good as well, but may not be that consistent in general and did not make it in the top 20 mean AUC list. Some networks though, like 2-2, 2-4, 3-4, 3-5, 3, 8, 2-2-4 etc are of special interest since they have featured in both the list of mean and max top 20 AUC. Even though the top 20 maximum AUC list is dominated by architectures with growth rate 24, in top 20 architectures based on mean AUC that is not the case. So still unable to select the best growth rate for the further evaluation. Hence further analysis is done for the best growth rate on a different view of the data.

Optimal growth rate analysis

It is also important to find out the best growth rate for each of the architectures (layers per block). In the previous test each architectures were evaluated for growth rates: 6, 12, 18, 24, 30 and 36. For each architectures the growth rate for which the best mean AUC and best maximum AUC is recorded are displayed in the graph below (figure 5.3)

From figure 5.3 it is observed that the growth rate for which each architecture have best mean and for which it has maximum AUC, might not always be same. For this purpose, histogram of best performing growth rates obtained from mean and max AUC analysis is displayed in the figure 5.4 below:

Discussion

It is evident from figure 5.4b, based on the max AUC, there is no strong trend, its very random and inconclusive. Which is not very surprising given its just one run. With so many parameters and initialization and random dropouts involved, the network weights might learn very differently in spite of being trained in a same condition, resulting in a very different decision boundary. In figure 5.4a it is visible that the contribution of growth rate 6 and growth rate 36 is really less, so probably they are too thin or too thick for the data. From figure 5.4 above it is safe to

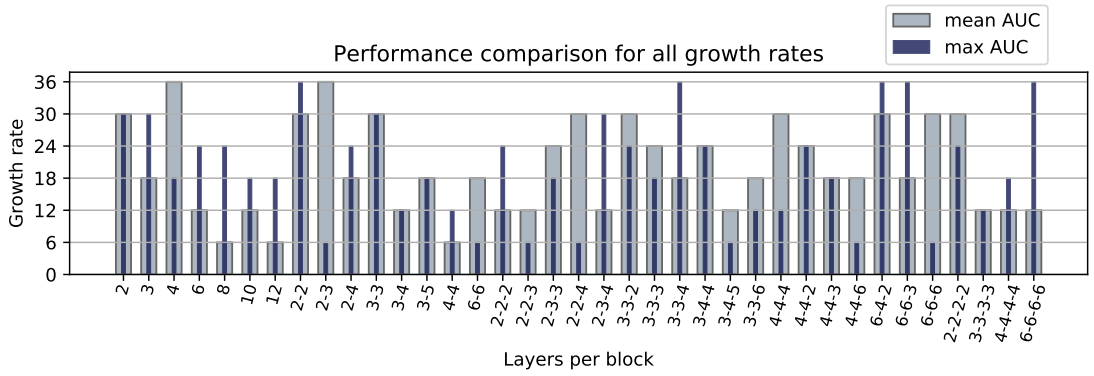
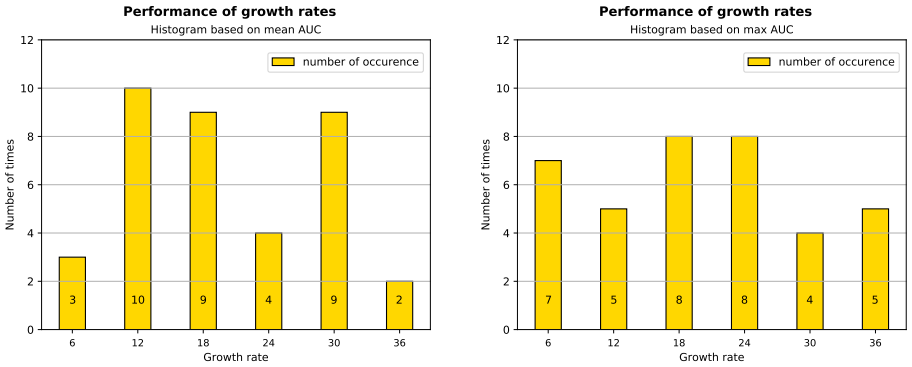


Figure 5.3: Best growth rate for each architectures are displayed based on mean AUC and max AUC. The grey column represents which growth rate ranked highest in mean AUC for the architecture, and the dark blue bar represents the growth rate for which the maximum AUC was recorded.

assume that growth rate 18 is very good performer in both analysis. Which also makes sense since it is neither too thin nor too thick. Because this conclusion is based on just 5 evaluations of each architectures, it make sense to experiment with other growth rates(except 6,36) as well for finer evaluation.



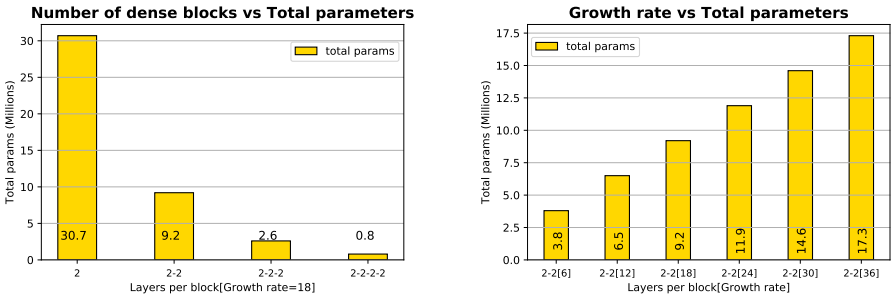
(a) Histogram of growth rates depending on mean AUC

(b) Histogram of growth rates depending on max AUC

Figure 5.4: Cumulative growth rate analysis (histograms)

5.1.3 Total parameters analysis

It might be surprising but the single dense block networks have **most** parameters. This is also because of the flatten pooling that is used here in this work instead of global average pooling 2D. And four block networks have the least total and trainable parameters. This is also supported by the theory in the paper [15] the more layers the network has the number of parameters gets lesser. How ever as the number of dense blocks keeps getting higher the non-trainable parameters also gets higher. So 4 blocks dense net has most number of non-trainable parameters. For the visualization of the comparison 2, 2-2, 2-2-2, 2-2-2-2 layers per block DenseNet's parameter sizes are compared below, all recorded for growth rate 18.



(a) Parameters vs architectures

(b) Parameters vs growth rates

Figure 5.5: Total number of parameters analysis

In figure 5.5b it is shown that with growth rate increase the total parameter size also increases. For this purpose the parameters for all the growth rates compared for architecture 2-2.

5.1.4 Standard deviation across blocks

Another trend was observed that the standard deviation varies more as the number of dense blocks increase. So the standard deviation values for all the readings are collected for 1, 2, 3, 4 number of dense blocks (nb_dense_blocks) separately and their average values are presented in the table below.

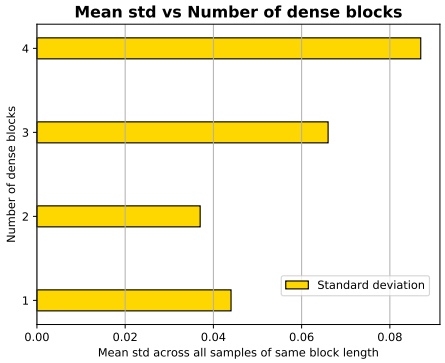


Figure 5.6: Average standard deviation on AUC across networks with same number of dense blocks, e.g., 2-2 and 3-3 has 2 dense blocks, 2-2-2, 3-4-5 has 3 dense blocks etc.

Even though the number of sample size differs a lot, 42, 48, 108, 24 samples for dense blocks 1, 2, 3, 4 respectively, it is observed that standard deviation of AUC is higher for dense block 3 and 4. It is probably because the blocks 3 and 4 networks have much lesser parameters than the 1 and 2 number of dense blocks. It is clear from this analysis that the two layer architecture is the best.

At this point, the best possible values of the growth rate and layers per block have been obtained. But the search space for the architectures are still big and needs to shortened further and basically chose up to 3 networks so that further evaluations can be done.

5.1.5 Finer grid search analysis

From the first analysis the top 5 network based on mean AUC of 5 trials (figure 5.1) and top 5 network obtaining maximum AUC (figure 5.2) are further evaluated for **20** evaluations each, after training from scratch. This 10 architectures will be referred as top 10 architectures in following analysis. All other test conditions remain the same. Results after 20 times evaluation is expected to be more dependable than 5 trials. In the chart 5.7 the mean AUC of 20 evaluations and it's standard deviation is displayed in yellow bars and blue lines respectively. While in red displayed the maximum AUC obtained in 20 evaluations.

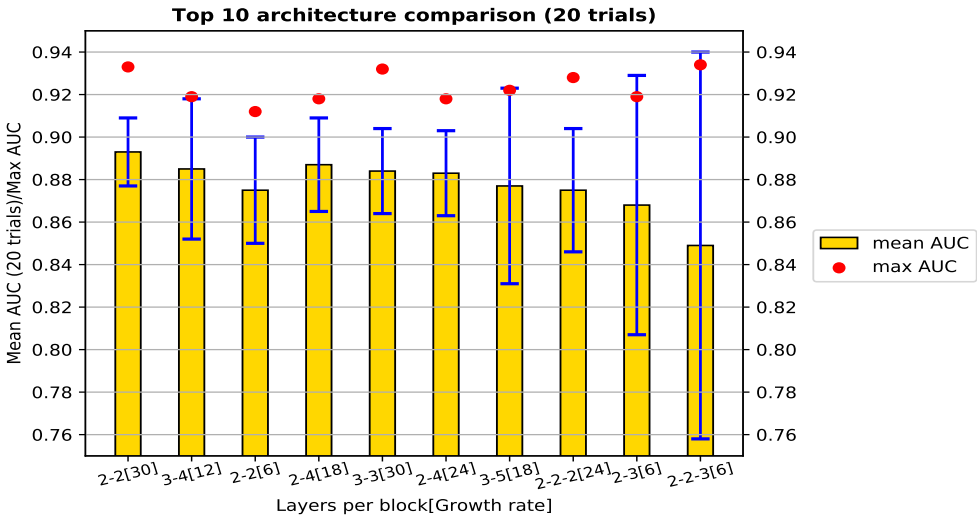


Figure 5.7: Finer search with top 10 architectures

Same data is displayed in table 5.2 but in decreasing order of the best mean AUC.

Table 5.2: Finer search results for 20 evaluation of previous top 10 architectures. Sorted according to mean AUC.

Layers	Growth rate	Mean AUC	Std	Max AUC
2-2	30	0.893	0.016	0.933
3-4	12	0.885	0.033	0.919
2-2	6	0.875	0.025	0.912
2-4	18	0.887	0.022	0.918
3-3	30	0.884	0.02	0.932
2-4	24	0.883	0.02	0.918
3-5	18	0.877	0.046	0.922
2-2-2	24	0.875	0.029	0.928
2-3	6	0.868	0.061	0.919
2-2-3	6	0.849	0.091	0.934

Discussion

It is observed that in 20 evaluations layers 2-2 with growth rate 30 is the best result both in terms of lowest standard of deviation and highest mean AUC. As it happens its also second highest in terms of the maximum AUC 0.933 just behind 0.934 from 2-2-3. 3-4, 2-4 networks are also performing well in terms of mean AUC. Their results are very close as well, so just evaluating 3-4 network for the finer analysis. 2-2-3 layers is interesting though, it has the highest standard deviation but 2 or 3 very good AUC scores too. So it needs to be further looked into.

In figure 5.8 the five number summary(min, first quartile Q1, median, third quartile Q3, max) is compared for architecture 2-2 and 2-2-3. For architecture 2-2-3, 3 AUC readings are detected as outliers out of 20 trials at 0.666, 0.608, 0.656 AUC. The outliers are affecting the overall mean auc for 2-2-3 network, also causing big standard deviation. This outliers are probably caused by training getting stuck in local minima or similar. 2-2 is found to be more consistent, it has no outliers. It is believed that consistency is desirable for a network. Hence 2-2-3 network is ruled out of contention for the best network, because of it's lack of consistency. though the overall concept of terming few prediction accuracies as outlier can be debatable. It is the nature of the network. At least, it is clear that three prediction accuracies are way of than other 17 predictions.

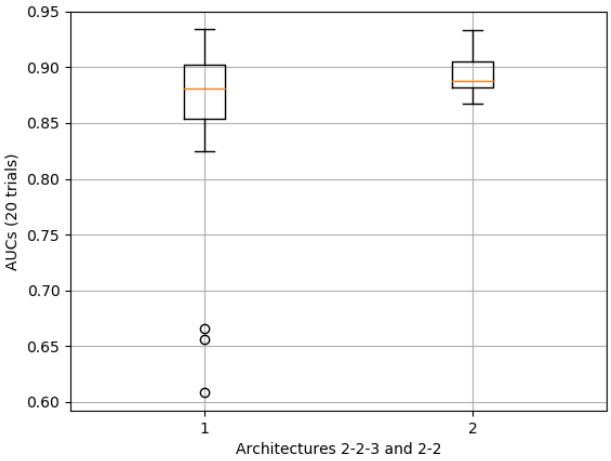


Figure 5.8: Box and whisker plot representation of AUC predictions of 2-2-3 and 2-2 architectures. First boxplot is for 2-2-3.

Five number summary

Also the brief introduction to the five number summary is as follows. **The minimum** is the smallest datum in the dataset. **The first quartile** is chosen so that 25% data points are lesser than it, similarly **the median** is the middle point (50%) of the dataset, and **the third quartile** is the point where 75% data falls below it. Lastly **the maximum** is simply the highest datum in the data set. All together this five criteria represents different aspects of any distribution.

Box and whisker plot interpretation

Box and whisker plot [12], where the box represent the interquartile range between first quartile 25% (Q1) and third quartile 75% (Q3) for the data range, and the orange line is for the median and the extended whiskers display the range of the maximum and minimum data points in the distribution. The interquartile range denoted by IQR, is the difference between Q3 and Q1. Outliers are the data points that reside outside the stretch of [(Q1-1.5*IQR), (Q3+1.5*IQR)]. The fraction 1.5 here is the default value in 'matplotlib' and also used in this work for all the evaluations. So if the data points lie outside the aforementioned range, then those are displayed as little dots or hollow circles in the plot beyond the whiskers.

5.1.6 Number of filter analysis

Initial number of filter (parameter name 'nb_filter') values 8, 16, 32, 64 are being evaluated here. Also a comparison between mean prediction AUCs obtained for growth rate 18 and 30 is done under this analysis. Ten evaluations of each test cases has been done.

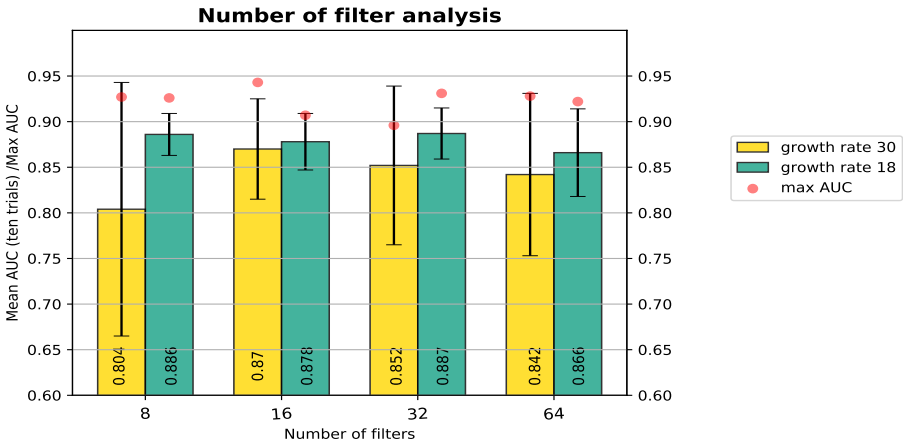


Figure 5.9: Number of filter analysis for different growth rates

Discussion

figure 5.9 shows that the number of filter 16 works better than others for growth rate 30 and also happen to have the Maximum AUC recorded and lowest standard deviation as well. It seems for thinner networks (lower growth rates), change in number of filter value matters lesser than the thicker networks. Since for 18 growth rate the mean AUC for all the values of 'nb_filter' are very close. But there is lot of difference for growth rate 30 for different rates. However both the networks were found to work pretty good with number of filter value of 16. This could have been further evaluated with more growth rates, but for this work it is concluded with 16 as our best nb_filter value.

5.1.7 DenseNet dropout probability analysis

As usual ten evaluations done for each dropout probabilities displayed in figure 5.10. Mean AUC is best for dropout 0.4. Maximum AUC is highest for dropout 0.5. Now it is no surprise that with 0, 0.1 and 0.7 dropouts the results are not the best, because it's either too less or too much regularization. But it is bit unexpected to have dropout probability 0.3 and 0.5 performing low. probabilities 0.2, 0.4 are chosen as the best dropout rate values for future evaluations because they have comparatively better max AUC and mean AUC.

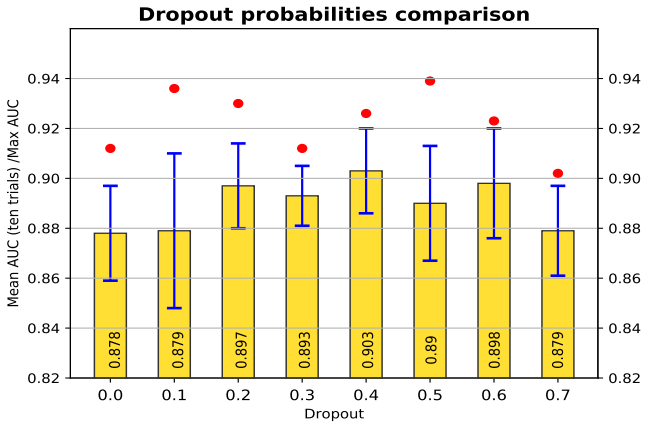


Figure 5.10: DenseNet dropout probability analysis.

5.1.8 Bottleneck and compression analysis

The evaluation results for the bottleneck and compression are displayed in the figure 5.11.

Discussion

The use of compression really makes the model much more compact without losing the effectiveness. From figure 5.12 it is observed that without compression the parameter size is 20.1 Millions, after using compression of 0.7 the mean AUC is still as good but the total parameter size has become 12.1 millions. Use of

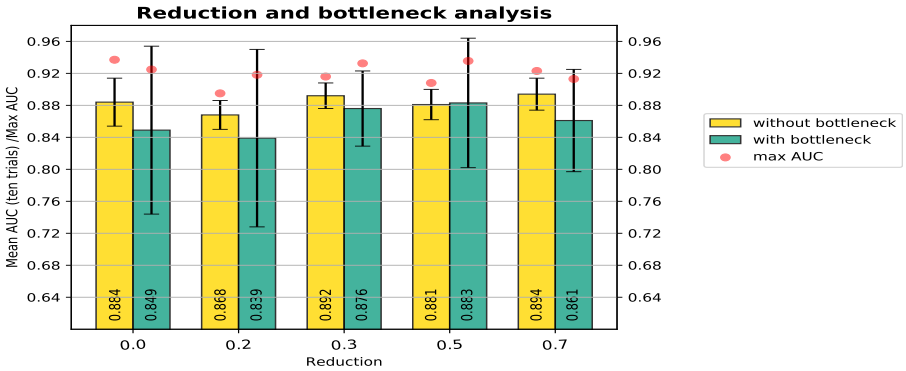


Figure 5.11: Evaluation of compression and bottleneck and mean AUC(across 10 trials). The max AUC obtained by one of the trial is also displayed separately for experiment with bottleneck and without bottleneck.

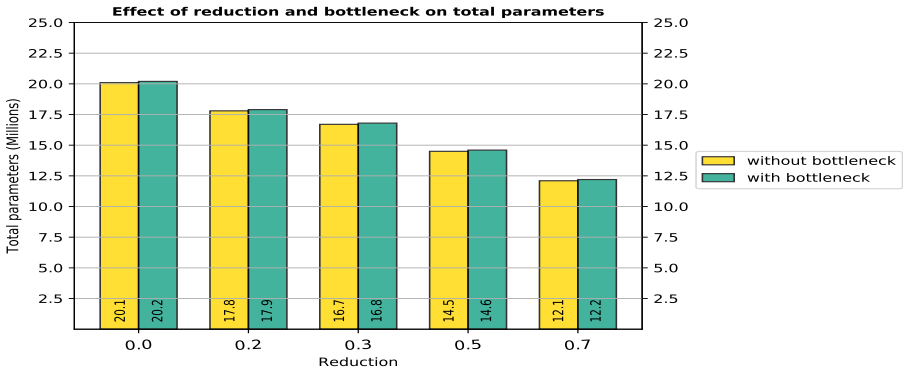


Figure 5.12: The total parameters are varying with reduction. It also gets affected by use of bottleneck, how ever that is very minimal.

bottleneck does in-fact increase the parameters by little more than 0.1 million, but it does not improve the results at all. So this is a unexpected behavior. Bottleneck also has much higher standard deviation. Theoretically it is suppose to help making the model more compact. But over all effect on the data is observed to be adverse. But the max auc values still belong to the bottleneck layers 3 out of 5 times. Other two times also its close to highest. That's perhaps interesting to note. So the best reduction ratio chosen, based on the mean AUC are 0.7 and 0.3. Bottleneck will not be enabled for further evaluations.

5.1.9 Fully connected layer dropout analysis

In the diagram 5.13 the hyperparameters associated to the decision network part are displayed.

The dropout probability connected to the first FC layer of the Siamese part is evaluated here, shown as d in 5.13. The search space evaluated for the d is as follows: [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7].

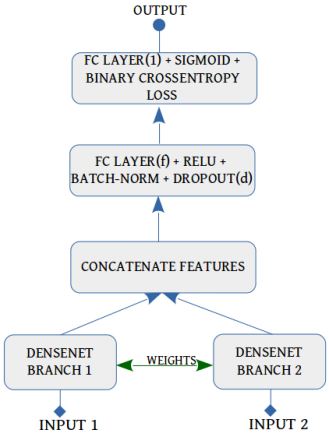


Figure 5.13: Hyperparameters in DenseNet-Siamese architecture

Discussion

From figure 5.14 it can be observed that for dropout probability 0.5 the network had the highest mean AUC along with dropout probability 0.7. The mean AUCs for 0.3, 0.4 is lower than expected and for 0.7 is much higher. The best values for dropout probability chosen for further evaluation are 0.5 and 0.7 both.

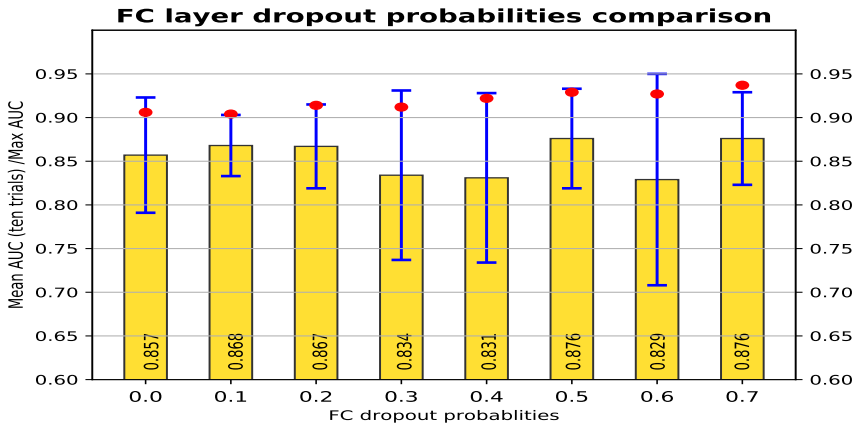


Figure 5.14: Optimal dropout (d) probabilities for FC layer analysis

5.1.10 Fully connected layer output size analysis

The FC layer output size is denoted by f in figure 5.13. The fully-connected layers are initialized (kernel) with He normal initialization.

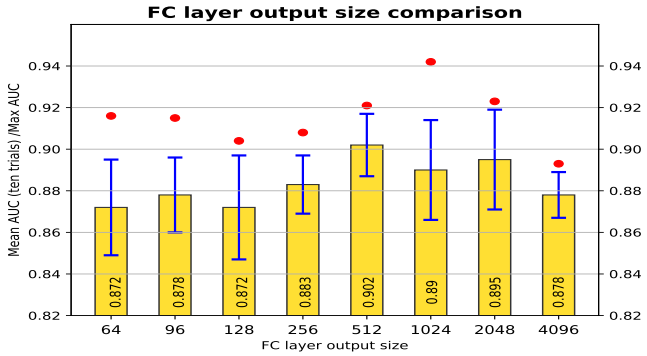


Figure 5.15: Optimal FC filter size (f) analysis.

Discussion

Whole search space is shown in figure 5.15. The output (f) value 512 yielded the best mean AUC (from the figure 5.15). Though the max AUC obtained by the f=1024 and f=2048 is also scoring very good. However on this value of f the number of total parameters for the whole network depends as the whole feature map of each DenseNet branch are flattened using flatten then concatenated. Hence multiplied by 2 (since 2 branches). This concatenated feature map is then multiplied by the f size when they gets connected. For example one DenseNet branch has feature map has size=n, concatenated feature map has size=2n. Concatenated features gets connected to FC network with output size f results in 2n*f parameters increase, and produces output size = f. In other words the smallest f size which gives good performance, is better since it keeps the computations smaller. So f=512 is chosen.

5.1.11 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 [19].

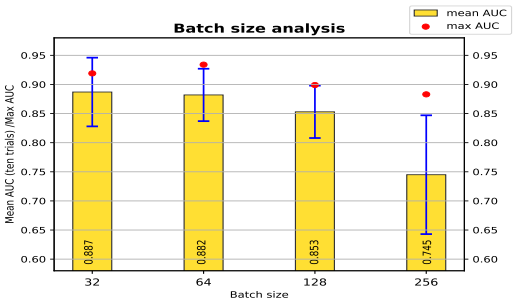


Figure 5.16: DenseNet-Siamese optimal batch size analysis.

Discussion

From the figure 5.16 it is observed that the larger the batch size gets the prediction accuracy on the test data gets worse. Mean prediction AUC for batch size 32 and 64 are very close. The max AUC score for batch size 64 is much higher than 32 and it will train faster too, that is why it is chosen as the best value.

5.1.12 Learning rate and optimizer analysis

During the training, in backpropagation step, the analytic gradient is computed which is used to update the parameters of the network (inspired by [31]. 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 [13]. It compares the loss function to a hill and randomly initializing the network weights to a 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 [7] 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 available e.g., with momentum, Nesterov etc), this optimizer is not evaluated in this work in favor of more theoretically advanced optimizers. In **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 advanced optimizers. **RMSprop** try to compensate the aggressive monotonically decreasing learning rate from Adagrad by introducing the moving average of squared gradient. **Adam** can be seen as RMSprop with momentum. **Nadam** incorporates Nesterov momentum into Adam. **Adamax** is a variant of Adam which uses infinity norm. **Adadelta** is like Adagrad with moving window of gradient updates.

Optimal learning rate selection is very important for effective learning. However, optimal learning rate varies optimizer to optimizer, hence for learning rate and optimizer a very fine grained search is performed here The search space contains total 20 different learning rates and five optimizers. Each optimizers were evaluated for all 20 learning rates, that makes 100 network configurations which were trained 10 times from scratch for the evaluation and compared on the prediction accuracy (AUC) and std as usual. The search space is presented in table 5.3:

Table 5.3: The search space for learning rate and optimizer best hyperparameters. Since both are related they need to be evaluated together.

Learning rate
0.1, 0.5, 1.0 (only for Adadelta)
0.01, 0.02, 0.03, 0.05, 0.07
0.001, 0.002, 0.003, 0.005, 0.007
0.0001, 0.0002, 0.0003, 0.0005, 0.0007
0.00001, 0.00002, 0.00003, 0.00005, 0.00007
Optimizers
Adam, Nadam, Adamax, RMSprop, Adadelta

The evaluation results are presented in figure 5.17 where the results for different

learning rate are compared for each of the optimizers. In figure 5.18 different representation of the evaluation result is presented which offers the comparative view for each of the optimizers for specific learning rate.

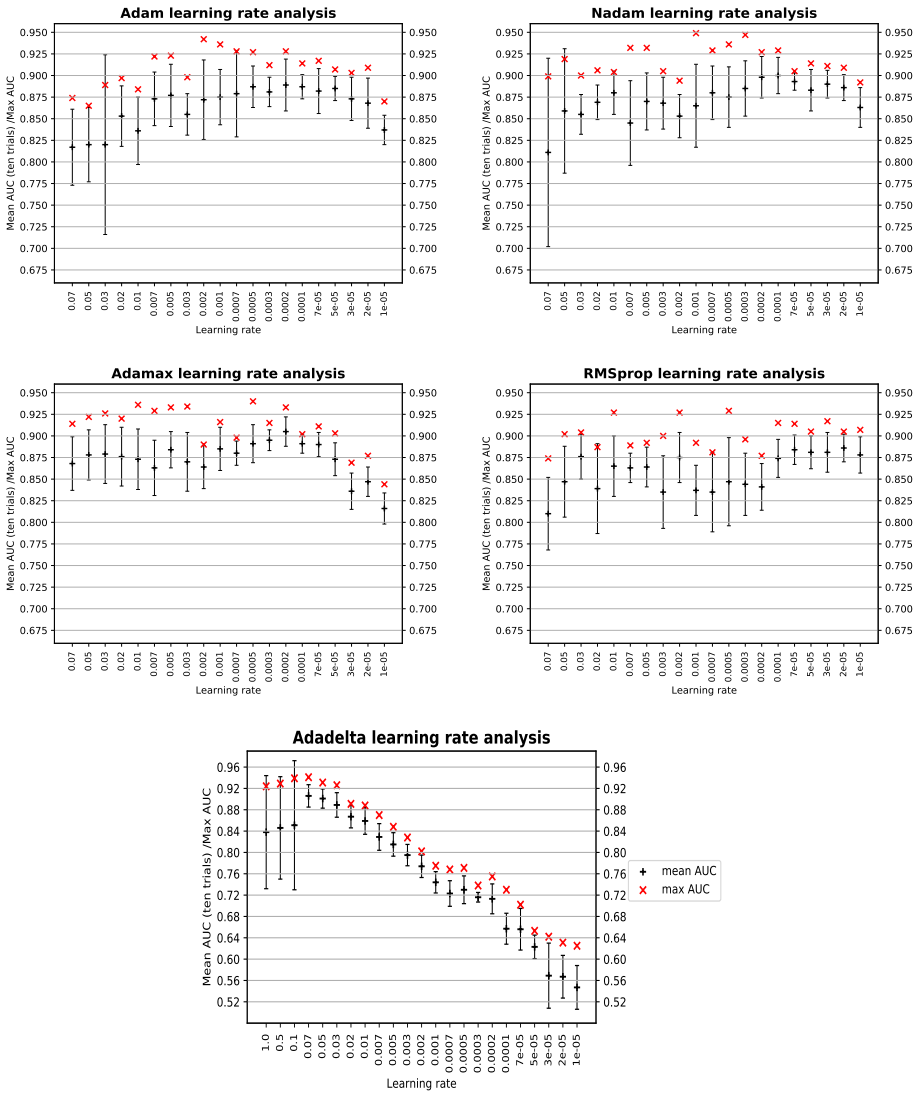


Figure 5.17: In order, (a)Adam, (b)Nadam, (c)Adamax, (d)RMSprop, (e)Adadelta learning rate analysis

Discussion

Overall Adadelta optimizer with learning rate 0.07 has the highest mean AUC

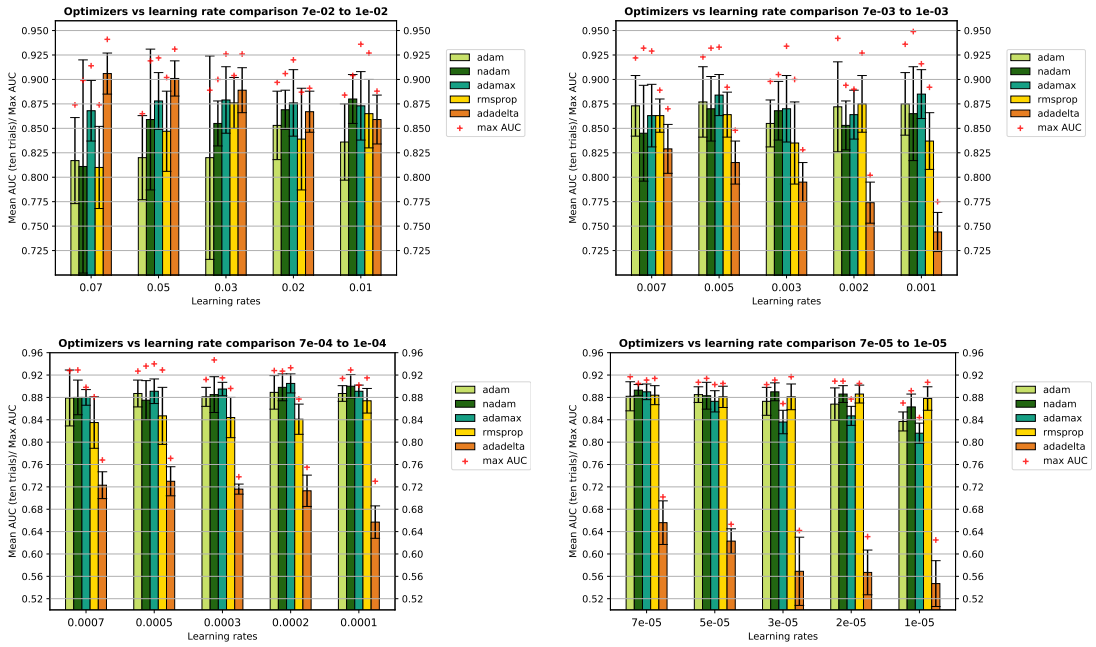


Figure 5.18: Comparison of different optimizers across different learning rates. For more effective comparison the total evaluation is visualized 5 learning rate at a time. Mean AUC obtained for all five optimizers for specific learning rate are grouped together, visualized using bars of different colors. Each two figures in a row have same limits for comparison.

0.906. Which is closely followed by Adamax and Nadam. Adam and RMSprop are slightly behind. The best mean AUC results are displayed in table 5.4. So there is no fixed learning rate for which all the optimizer works best, which is expected as well. For Adamax, Adam, Nadam learning rate 0.0002 works very good, for RMSprop learning rate 2E-5 works better. For Adadelta, as the learning rate drops the performance also drops significantly. For Adadelta the best learning rate is at the boundary condition (0.07) of the evaluation range. So further analysis needs to be done for higher learning rates like 0.1, 0.5, 1.0. In Keras the default learning rate for Adadelta in Keras is 1.0 so it is not really surprising.

Table 5.4: The learning rate for which the optimizers recorded it's best mean AUC.

Optimizer	Learning rate	Mean AUC(10 trials)
Adadelta	0.07	0.906
Adamax	0.0002	0.905
Nadam	0.0001	0.9
Adam	0.0002	0.889
RMSprop	0.00002	0.886

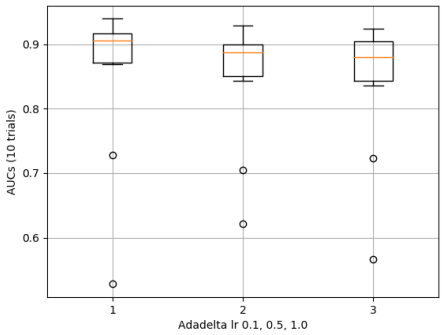


Figure 5.19: Adadelta evaluation for high learning rates. Two of the AUCs out of ten trials for each of the three configurations are displayed as the outlier in the Boxplot. In each of the cases the network did not train well and possibly remained stuck in local minima. The Boxplot displays statistical five number summary for the 10 AUCs recorded for each of the configurations.

From figure 5.19 it is observed that each of the trials with learning rate 0.1, 0.5 and 1.0 somehow had 2 AUCs out of 10, which are far lower than others. Which are displayed in the boxplot as the outliers. After discarding the outliers the mean AUC of the trials would drastically improve and displayed in the following table 5.5. It is however debatable, not fair to calculate mean AUC discarding some runs. In any case best learning rate 0.07 is chosen for Adadelta as it is found to be more dependable and high scoring too.

Table 5.5: Optimizer Adadelta evaluation with high learning rates. Displayed results are after filtering the outlier cases.

Learning rate	0.07	0.1	0.5	1.0
Mean AUC	0.906	0.906	0.891	0.887

5.1.13 Final grid search

The search for the best hyperparameters are done separately so far, now all the best performing hyperparameter values are put together as part of this final grid search. If the hyperparameters that were searched separately, did not have any interdependency (e.g., Learning rate and optimizer has), then the overall prediction accuracy should improve when all the best hyperparameter values are used together.

Hyper-parameters for final grid search

The search space for this final grid search has been narrowed down manifold by doing the individual or focused parameter search before. The remaining search space for the final search is described below in table 5.6.

Table 5.6: Best hyperparameter values for final evaluation.

Name	Value	Name	Value	Name	Value
Number of filter	16	Layers	2-2, 3-4	Growth rate	12, 18, 30
DenseNet dropout	0.2, 0.4	Compression	0.3, 0.7	Bottleneck	'False'
FC output	512	FC dropout	0.5, 0.7	Pooling	'flatten'
Batch size	64	Optimizer & learning rate	'Adadelta' & 0.07, 'Adamax' & 0.0002, 'Nadam' & 0.0001		

From the table 5.6, it is observable that three different optimizers were evaluated with their corresponding learning rates. So basically each test case were evaluated for each optimizer. Similarly for other hyperparameters, which have more than one values in the search space, the overall search cases are multiplied by the number of those many values. Here total of 48 x 3 (for optimizers) = 144 dimensional search space is being searched for the final evaluation i.e 144 network configurations. Each of them will be trained from scratch 10 times and the network configuration with best mean AUC for test data prediction, is considered to be the best network. And the corresponding hyperparameter values will be the best hyperparameter values for DenseNet Siamese network.

Table 5.7: Best hyperparameter values for the DenseNet Siamese, obtained from the final grid search. In decreasing order of their mean prediction accuracy on the test data. The results are very close, hence top 5 results are displayed instead of just one.

Config alias	Epochs	Learning rate	Optimizer	Layers	Growth rate	DenseNet dropout	Compression	Mean AUC	Std	Max AUC
'Config 1'	14	0.07	'Adadelta'	2-2	30	0.4	0.3	0.921	0.016	0.95
'Config 2'	15	0.0002	'Adamax'	2-2	18	0.4	0.7	0.918	0.009	0.935
'Config 3'	14	0.07	'Adadelta'	3-4	12	0.4	0.7	0.915	0.019	0.94
'Config 4'	14	0.07	'Adadelta'	2-2	18	0.4	0.3	0.913	0.012	0.927
'Config 5'	13	0.07	'Adadelta'	2-2	12	0.2	0.7	0.912	0.011	0.932

The hyperparameters which had multiple values in search space, the best values are mentioned in table 5.7. Rest of the hyperparameter values are fixed to what was displayed in table 5.6. From table 5.7 the 'Config aliases' are used to refer to the network structure in the following figures, for comparing the results.

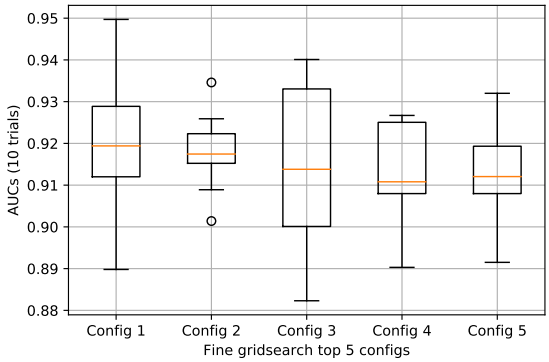


Figure 5.20: Top five DenseNet Siamese network each evaluated 10 times from scratch. For most effectively compare the network statistical analysis is done for all ten AUC prediction results for each of the network and visualized using Boxplot

Discussion

The best result reported in Valdenegro et al. [36] is **0.91** AUC for binary class prediction and **0.894** for the score prediction, for the same data used in this thesis. However, In this work only score prediction is done. DenseNet Siamese network able to record highest mean AUC in ten trials **0.921**, with standard deviation across the trials of 0.016 and maximum AUC of 0.950. Which is higher than the state of the art score prediction. The best performance was recorded with Adadelta optimizer and learning rate 0.07.

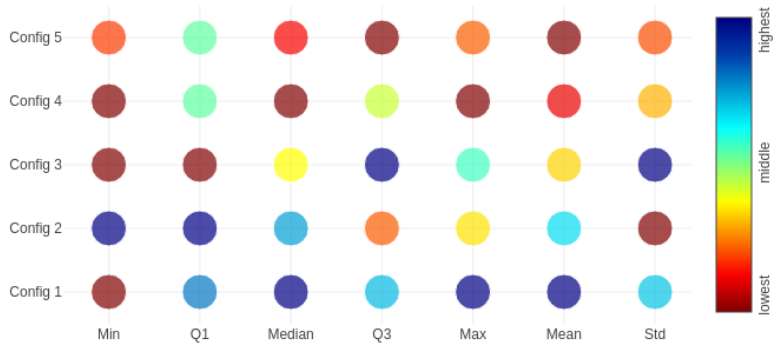


Figure 5.21: Statistical analysis on top results from final grid search visualized in bubbles with color codes which enable the qualitative analysis among different configurations.

The mean AUC result for the top configurations are very close and it is hard to declare a runaway winner as the highest network has mean AUC of 0.921 and std of 0.016, second best has mean AUC of 0.918 with std 0.009.

It is interesting to note that the dropout values which are giving the best results are high 0.7 for the Siamese side and 0.4 (only for 'Config 5' it is 0.2) for the DenseNet side. High dropout is enabling the network to be good in generalization. The training/validation accuracy is could reach very high, still the generalization on test data is high.

The statistical comparison of the evaluations which yielded top 5 mean AUC are displayed in figure 5.20. Each of the 5 network configuration were evaluated for 10 times and the prediction accuracies are compared quantitatively based on the box and whisker plot representation.

Interestingly enough, for 'Config 2', two points are displayed to lie out side the boxes. This supposed outliers, both in higher and lower end of the distribution range, are not really outliers, it just so happens that this two values are bit higher and lower than the rest 8 Auc values, which have only standard deviation of only 0.05. In fact this shows that this result is the most consistent one, it is also the only top configuration that uses 'Adamax' optimizer and as seen from the 5.7 the

compression ratio is 0.7. In comparison to the 'Config 1' the compression ratio used for 'Config 2' is higher. As previously seen in figure 5.12 the total parameters for the networks are much lesser for the 'Config 2' than 'Config 1'. So overall this result for 'Config 2' is also very good.

And in figure 5.21 multiple statistical data is displayed in bubbles whose color indicates the scale of the value. Lowest values indicated by brown or shade of red and as it gets higher the color gets orange, yellow, green, cyan to blue and darker blue. Each column may have data in different ranges, so for most effective qualitative comparison each columns, which are one of the statistical measure, the color maps normalized for each of the column. It is done automatically, by assigning the lowest value of the 5 results for each metric to the lowest color map and highest to the highest color map. So in this one image 5.21 results from 50 evaluations are summarized. That is 10 evaluations for each of the 5 network configurations. The visualization displays statistical five number summary (min, Q1, median, Q3, max) and mean and standard deviation are computed from each of the 10 evaluations for each configuration, which signifies how each network performs.

This visualization is intended for rough qualitative analysis among the top configurations. It is clear that the 'Config 1' network outperforms other networks, because overall it has most 'bubbles' which are blue or dark blue, which means high values. Except the min value is very low, it's displayed in brown. Similarly,'Config 2' is also very good. It has the lowest standard of deviation. Apart from this configuration (with 'Adamax'), other four top configurations are using 'Adadelta'. 'Nadam' has got the best mean AUC of 0.912, which is same as the 'Config 5'. In figure 5.22 the Receiver operating characteristic (ROC) curve is presented for the best network 'Config 1'. The ROC curve is based on the evaluation which had maximum accuracy out of 10 runs for 'Config 1'. With this results the hyperparameter search for DenseNet-Siamese comes to the conclusion.

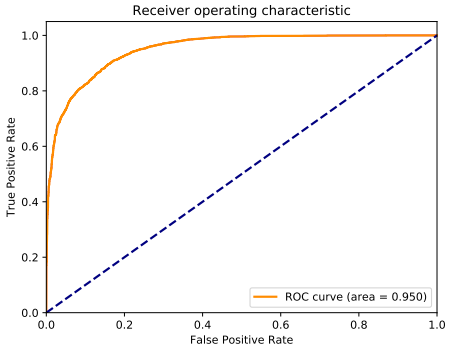


Figure 5.22: The area under curve is visualized for the best network 'Config 1'.

5.2 DenseNet Two-Channel

In order to get the best result from DenseNet two-channel **(DTC)** network for the dataset used in this work, best hyperparameter values needs to be found out. In the following section how the best hyperparameter search for DTC is conducted is described. Also the result of the DTC using the best parameter values are evaluated and the results are visualized for easier interpretation.

5.2.1 Hyperparameters to be evaluated

Overall hyperparameters of DTC can be divided into divided into two parts as follows:

Hyperparameters of DenseNet

This important hyperparameters were already mentioned in the section 5.1.1, during the best hyperparameter search for DenseNet-Siamese. While the structure and the intended functional usage are different the DenseNet two-channel and DenseNet-Siamese both depend on the basic feature extraction capability of the network. Hence there are some common knowledge which can be extracted from the hyperparameter search in previous section (DenseNet-Siamese network). But the main hyperparameters such as growth rate, layers per block, number of filters, compression and bottleneck still needs to be searched again, as the network structure is different.

Other hyperparameters

Apart from the DenseNet there are other general hyperparameters such as learning rate, batch size for training and optimizer.

5.2.2 DenseNet growth rate and layers per block analysis

Similar to the previous analysis, once all the hyperparameters which needs to be evaluated are known, manual optimization takes place. After many trial and error cycles or randomly the parameter values are changed around to get a intuition on the possible ranges for the hyperparameter values to be evaluated. This step also ensures that the starting point is not very bad. 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. Growth rate 6, 36 excluded as they are too thin and too thick respectively (found from DenseNet-Siamese evaluation). Different dense block sizes of 1, 2, 3, 4. There are other parameters but number of dense block, growth rate and layers per block are three main parameters. 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 incorporate minimal regularization. Epochs are different for different architectures to ensure that the networks are able to train decently. Flatten is used as pooling at the end of the DenseNet, in place of global average pooling. Binary_crossentropy loss function with Sigmoid activation function used for the binary classification, this final layer acts as the binary classifier. This is ensured by including the top of DenseNet architecture (`include_top=True`). In all the cases the networks are being trained from scratch. Parameter `weights` value None ensures that no previously trained weights are used.

Growth rate and layers per block search setup summary

The overall search space is summarized in the section below.

Fixed hyperparameters

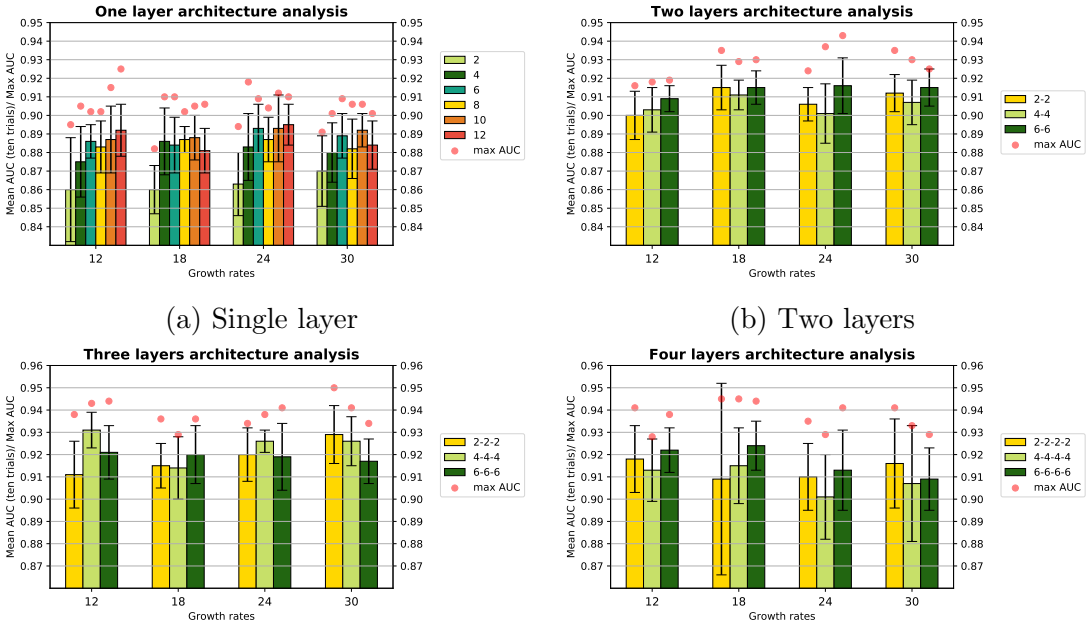
Other hyperparameters that are needed to instantiate the DenseNet are set to fixed values, in order to focus on the layers per block and growth rate parameters.

Table 5.8: Fixed hyperparameter values for the evaluation setup.

Name	Value	Name	Value	Name	Value
Number of filter	16	Subsample initial block	'True'	Weights	'None'
Dropout rate	0.2	Include top	'True'	Compression	0.5
Bottleneck	'False'	Pooling	'flatten'	Transition pooling	'max'
Optimizer	'adadelta'	Learning rate	0.07		

Varying hyperparameters

- **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



(a) Single layer

(b) Two layers

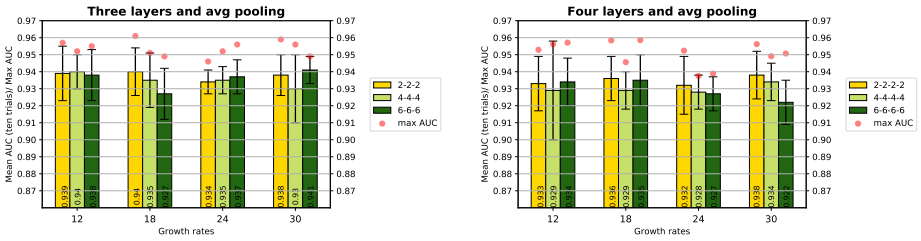
(c) Three layers

(d) Four layers

Figure 5.23: DenseNet two-channel layers per block and growth rate analysis. Unique layers per block is represented with a specific color code. Whole result is divided according to dense block numbers, for easier representation. Every two charts presented side by side have identical y-axis for comparison. All the results obtained for same growth rate were grouped together for each of the growth rate (12, 18, 24, 30) represented by x-axis.

Discussion

Each sub-figure shows mean AUC (on the test dataset) and standard deviation of 10 trials and maximum AUC. From figure 5.23 it is clear that 4 dense blocks and 3 dense blocks network performs better than the 1 and 2 block/s networks. The original paper [15] also used 3 and 4 blocks DenseNet for most of the evaluations. So this finding is expected. For growth rates though, no clear trend was observed, in fact, from sub-figure 5.23(c) it is observed that with increase in growth rate the '2-2-2' network performs better, while for '6-6-6' it mildly decreasing and similar for '4-4-4'. So no common best growth rate can not be determined for all the architectures, so it would be safe to evaluate for as many as possible. Growth rate 12, 18 and 30 will be evaluated for the final run.



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

Figure 5.24: DenseNet two-channel avg pooling analysis

5.2.3 Pooling analysis

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 [15] used global average pooling (`avg`) [24]. In the previous study `flatten` pooling was evaluated, for this analysis `avg` pooling is evaluated for 3 layer and 4 layer DenseNet blocks, since they were the best performing network in previous analysis. In figure 5.24 it is observed that between the three layer and four layers DenseNets, the former performs better. Overall the avg pooling is resulting in smaller total network parameters and also the better mean AUCs.

For a comparative analysis between the flatten and average pooling the mean AUCs are compared for each of the growth rate and for three layers DenseNet (2-2-2, 4-4-4, 6-6-6). Apart this layers and growth rate values, and pooling, other parameters remain same as the basic evaluation network 5.2.2.

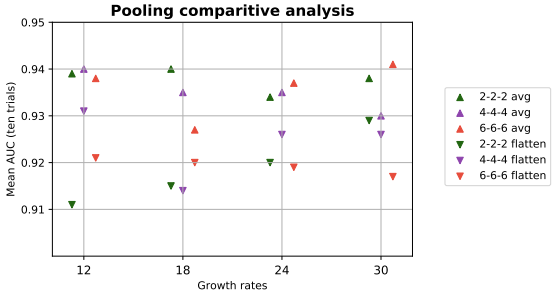


Figure 5.25: Average vs flatten pooling

Discussion

Here, the triangle up represents the mean AUC obtained using average pooling, triangle down represents flatten pooling. The results from the DenseNet architectures have been grouped together for each growth rates accordingly. So the x-axis of the graph shows growth rates. The y-axis represents the mean AUC obtained from 10 trials of each configurations. From figure 5.25 it is clearly seen that the average pooling produces better results than using flatten, in all the cases. Since the representation might be bit complex, the same data from figure 5.25 is displayed in tabular view (5.9) as follows.

Table 5.9: The average and flatten pooling comparison results.

Architecture	Growth rate	Flatten	Average
2-2-2	12	0.911	0.939
2-2-2	18	0.915	0.94
2-2-2	24	0.92	0.934
2-2-2	30	0.929	0.938
4-4-4	12	0.931	0.94
4-4-4	18	0.914	0.935
4-4-4	24	0.926	0.935
4-4-4	30	0.926	0.93
6-6-6	12	0.921	0.938
6-6-6	18	0.92	0.927
6-6-6	24	0.919	0.937
6-6-6	30	0.917	0.941

So the

Number of 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.

Discussion

With change in nb filter size the mean AUC varies a lot for higher growth rate such as 30, for growth rate 18 it does not vary so much. For growth rate 30 the nb

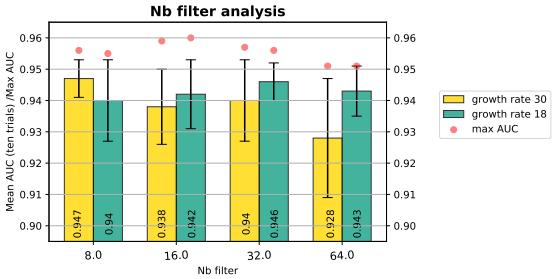


Figure 5.26: Nb filter size analysis analysis

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.

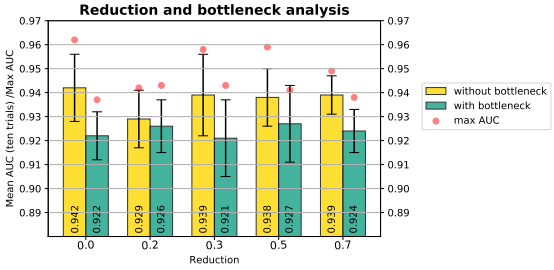


Figure 5.27: Reduction and bottleneck analysis

Discussion

- 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.

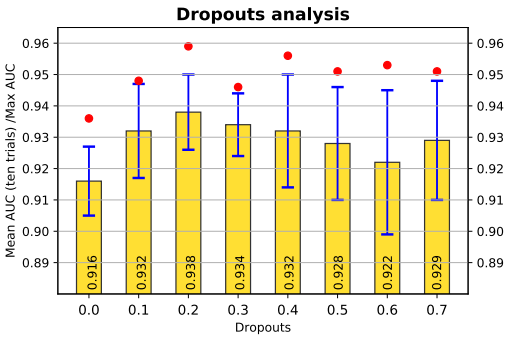


Figure 5.28: Dropouts analysis

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)

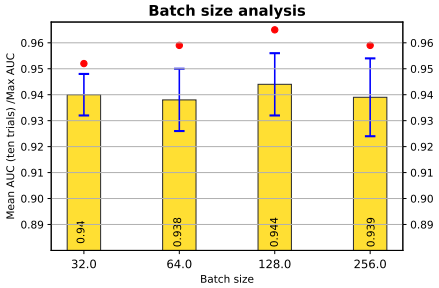


Figure 5.29: Batch size analysis

Discussion

From the figure 5.29 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.

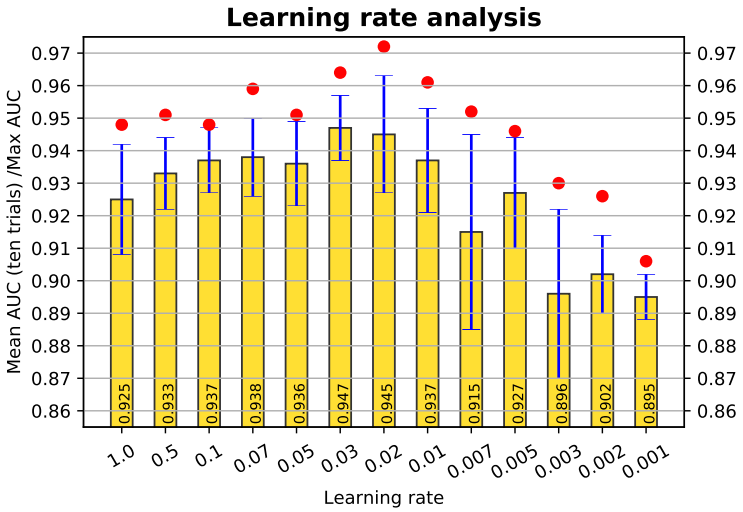


Figure 5.30: Learning rate analysis

Discussion

From figure 5.30 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.

hyperparameters 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 up to 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 consistent 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.
- 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 5.10. 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 5.10 the growth rate is displayed as Gr. and Reduction is displayed as R for space constraint.

Discussion

- 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.

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 5.10: Final grid search results

- In Valdenegro et al. [36] 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.

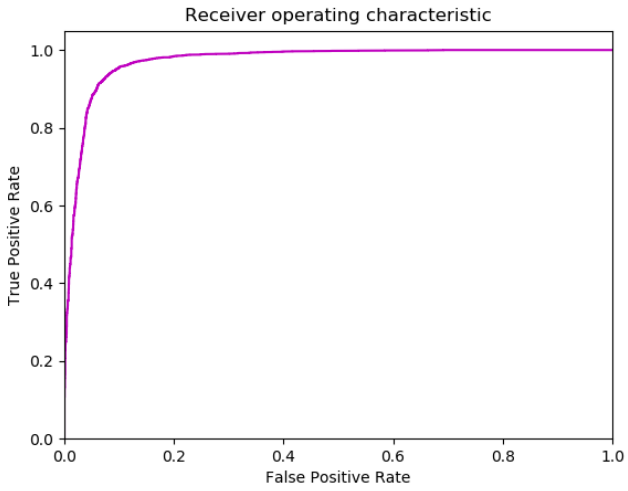


Figure 5.31: Roc AUC overall best result in all three architectures 0.973 AUC (Single run)

5.3 Contrastive loss

First target of the hyper parameter search is to finalize the main structure of the branches. Which has architecture like Conv(n, a x a)-Conv(n, a x a)-MP(2, 2)-Conv(2n, a x a)-Conv(2n, a x a)-MP(2, 2)-Conv(4n, a x a)-Conv(4n, a x a)-Conv(4n, a x a)-MP(2, 2)-Conv(8n, a x a)-Conv(8n, a x a)-Conv(8n, a x a)-MP(2, 2)-Conv(8n, a x a)-Conv(8n, a x a)-Conv(8n, a x a)-MP(2, 2)-FC(d). The FC layers could be repeated l times, when possible values of l are 1, 2 and 3. Also there could be batch normalization layers after all the FC layers. The batch normalization layer could be placed before the activation ReLU layer or after. Variables n, a, l, d are set with predefined set of values, from which the best performing combination will be selected for further evaluation. The overall coarse search space is displayed below,

5.3.1 Best hyperparameters search spaces

The hyper parameters of the network are as follows.

1. Conv filters(n)(8,16,32,64)

2. Kernel size(a)(3,5,7)
3. FC Layers (single, two, three)(l)
4. Initializers (He_normal, He_uniform, Glorot_uniform(default), Glorot_normal(Also called Xavier normal) RandomNormal.
5. FC filters (32, 64, 96, 256, 512, 1024, 2048)
6. Dropouts (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8)
7. Batch normalization (True or False)
8. Batch Size (32, 64, 128, 256, 512)
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)

In the above section the hyper parameters and the search space has been displayed. Overall search space is divided into smaller parts and been evaluated with a common starting network configuration, that we found to be working good after some manual tuning and many trials.

5.3.2 Flipped labels

Since contrastive loss returns projected distance, here close to zero means similarity and 1 means dissimilarity. Although, in our original data label 1 represents similarity between patches. Hence the labels for train, validation and test data here are all flipped. `new_label = 1 - old_label`

Conv filters analysis

The 'filters'[4] defines the number of output filters in each convolution layers. Now for all the 13 convolution layers in the network the filters size can be easily calculated from the first filter size. In previous section it has been shown that the filters for the Conv layers are n, n, 2n, 2n, 4n, 4n, 4n, 8n, 8n, 8n, 8n, 8n, 8n respectively. Here 4n means 4 times n obviously.

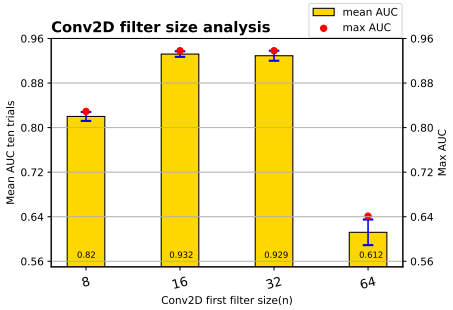


Figure 5.32: Conv filters analysis

Discussion From figure 5.32 the highest mean AUC is for 16. So for the best performing network is having the VGG branch with filters for the convolution layers as follows, 16, 16, 32, 32, 64, 64, 64, 128, 128, 128, 128, 128, 128. The performance for 32 is close too.

Kernel size analysis

The kernel size parameter defines the height and width of the 2 dimensional convolution window for all the Conv layers in the VGG network. table 5.3.2 it is clear that the kernel size 3 works so much better than 5 and 7 kernel size.

Rank	Kernel size	Mean AUC (10 trials)
1	3	0.932
2	5	0.799
3	7	0.481

FC units size analysis

The hyperparameter 'units' (d) in network determine the output size for a FC layer. Also how many FC(d) layers needs to be in the place needs to be determined.

From 5.33 single FC layer (l=1) in figure works best. But performance of two layers are also very close. 128 and 2048 has been included for the final grid search since they have the best results. Although conceptually this is bit unexpected to see both points close to the two extremes to work best, usually it should have been few points in the middle of the search space or near just one boundary but not both.

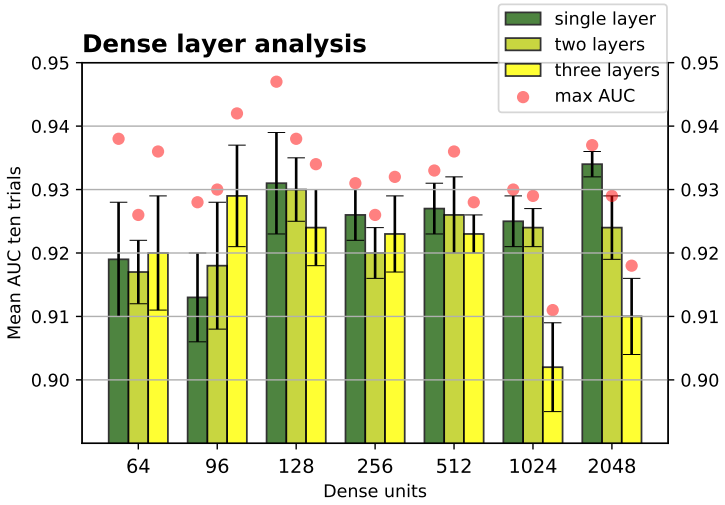


Figure 5.33: FC units and layers analysis

Initializer

Keras initializers[6] control or 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 layers and kernel initializers for the FC layers. The kernel initializers evaluated for the convolution layers (Conv) are He normal and uniform, Glorot normal and uniform and random normal. Initializers for Conv layers are chosen after some manual trials. For example it has been found that with RandomUniform (default settings) as Conv initializer the network is not converging at all. For the FC layers He normal and uniform and Glorot normal and uniform is used as initializers set to be evaluated. The bias initializer, for both Conv and FC layers, is used with default `'Zeros'` option. In keras for both Conv and FC the default kernel initializer is Glorot uniform. Popular Intuition is that Glorot or Xavier initialization works better with Sigmoid activation, while He uniform/normal works better with ReLU.In our case there are 13 Conv layers compared to only one or two FC layers, so the Conv 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, so it has been kept open for exploration.

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

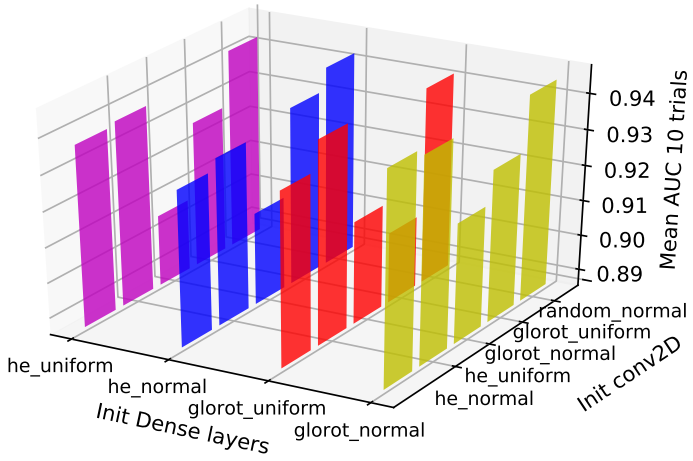


Figure 5.34: Comparison of performance of initializers for Conv and FC layers, in one axis the initializers for FC or Dense layers are displayed in unique colors, in the other axis the initializers for Conv layers are showed. Z-axis or height of the bars represents the mean AUC in ten trials obtained for each combination of Conv and FC initializers.

ten trials is shown. The z-axis values are clipped and starts from 0.89. This is to be able to show the differences in bar heights 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 random normal as the initializer for Conv layers over all good. Performance of the Glorot normal and uniform both as Conv initializer is comparatively worse than others. Performance of He initializers are over all quite good as a Conv initializer. As the FC layer initializer glorot normal is found to have performed the best. Over all performance wise the He normal as the Conv and glorot normal as the FC initializer has performed slightly better 0.943 than second best 0.941 with (He uniform, Glorot normal), and also (random normal, Glorot normal) both. The best mean AUC results are shown below. From table 5.11 it is

Rank	Init Conv	Init FC	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	Random normal	Glorot normal	0.941	0.004	0.946
4	He uniform	Glorot uniform	0.94	0.008	0.952
5	Random normal	He normal	0.939	0.005	0.946
6	Random normal	He uniform	0.939	0.005	0.948

Table 5.11: Kernel initializer top results

observed that there might be a trend that with random normal kernel initializers, the standard deviation is very low, but the max AUC values are consistently lower than the He counter parts. However all the top ranking results are very close in terms of mean AUC.

Batch normalization analysis

Small batch training is better than one by one training and also better than training the whole dataset all at once. Small batch training with batch normalization is advantageous because it converges faster than doing one by one. Batch normalization reduces the need for carefully tuning the initial weights, also to some extend limits the need for too much of regularizers such as dropouts. Concept of batch normalization was introduced by Ioffe and Szegedy in 2015 [18]. The authors were influenced by the idea from Lecun,1998b [22] and Wiesler and Ney, 2011 [38] that if inputs to a CNN are linearly transformed to have unit variance and zero mean, then the network will converge faster. The learning rates are generally kept comparatively lower because an outlier might cause big effect in already learned activations. As a result of keeping the inputs normalized the outlier cases also affects the overall learning process lesser. Hence Batch normalization should also enable the use of higher learning rates. The batch normalization layer is only applied after each fully-connected or dense layer. In this thesis, the batch normalization is evaluated three steps, firstly, without batch normalization. Then, with batch normalization, when adding the batch normalization layer after the dense layer but before the

Rank	Batch normalization	Mean AUC (10 trials)	STD	Max AUC
1	Without	0.932	0.005	0.938
2	Before activation ReLU	0.9	0.012	0.926
3	After activation ReLU	0.874	0.011	0.893

Table 5.12: Kernel size analysis

activation function 'ReLU'. Thirdly, after both, dense layer and activation 'ReLU'. Mean AUC comparisons are as follows (same epochs).

The findings are surprising, It has been noted during the training that the network can achieve higher training accuracy in the same number of epoch than without batch normalization. However the generalization performance on the test data is worse using batch normalization. With batch normalization cases were also tested for lesser and higher epochs than without, it still scores poorer results. May be this can be evaluated along with different batch sizes. May be also higher learning rates.

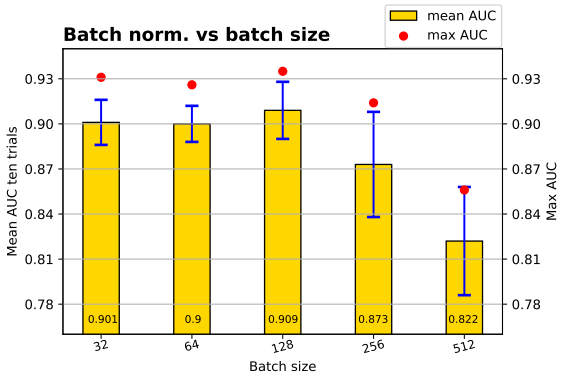


Figure 5.35: Evaluation of how batch normalization affects prediction with increase in batch size.

The batch size and batch normalization correlation analysis is done in figure 5.35. The idea was that the batch normalization performance might increase with increase in the batch size. The bigger the batch size is, the more it resemble the actual distribution that represents the whole data, this was one of the intuition. No strong evidence were found in this direction, though the mean AUC slightly peaked from batch size 32 to 128 and then sharply fell down. Since batch normalization

limits the effect of outliers, could it be the case that the apparent outliers contained discriminative features somehow. Because of sonar images have low signal to noise ratio, it could be that the perceived noise is actually signal, which gets somewhat filtered out by the batch normalization. However this is only speculation, no concrete analysis was done in this direction. But batch normalization layers are not used for further evaluation.

Dropouts analysis

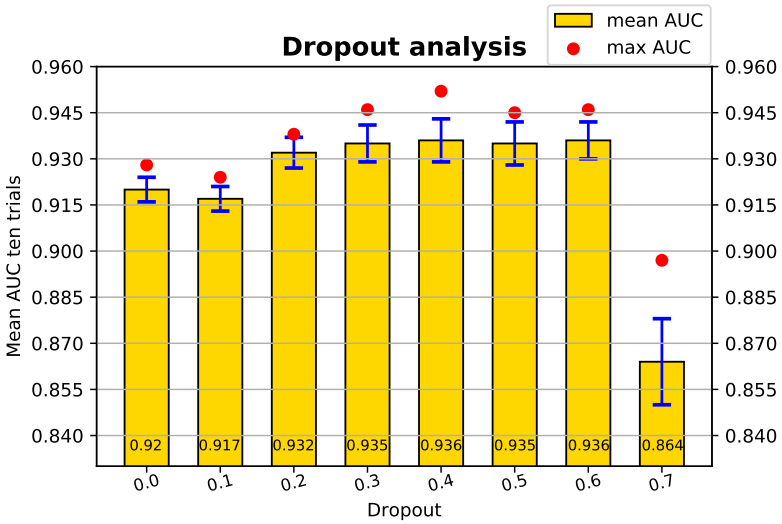


Figure 5.36: Dropouts analysis

From the figure 5.36 it is noticeable that the performance for dropout values 0.3, 0.4, 0.5 and 0.6 are very close and the higher than with too less dropouts like 0, 0.1. Also higher than high dropout value such as 0.7. 0.4 is chosen as the best value for the dropout, because it has highest max AUC value along with highest mean AUC. This result is very expected, based on the common findings.

Learning rate and optimizer

During the training, in backpropagation step, the analytic gradient is computed which is used to update the parameters of the network (inspired by [31]. 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 [13]. It compares the loss function to a hill and randomly initializing the network weights to a 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 [7] 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 theoretically advanced optimizers. **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 advanced 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.

Discussion From figure 5.37 it is found 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. Although 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 mean AUC and learning rate results are as follows: Adam-0.00001, Nadam-0.0002, Adadelta-0.005, Adamax-0.0007, RMSprop-0.0002

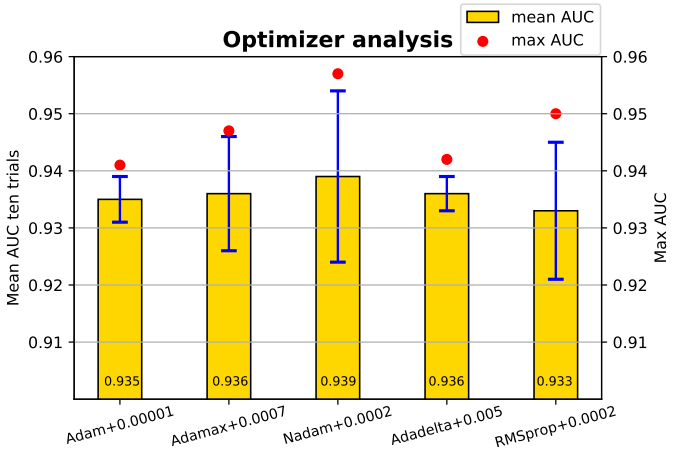


Figure 5.37: Search results of best optimizer and learning rate

Batch size analysis

If the batch size is too low then it takes more time and lower than a certain size network does not train well too.If the batch size is very big then it may train faster but might converge to sharp minimizers of the training function. This might result in poor generalization.

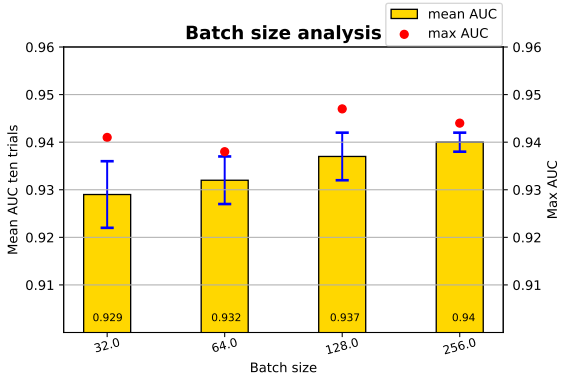


Figure 5.38: Batch size analysis

It is observed from the figure 5.38 that the test AUC somehow increases with the increase in batch size. Although it takes more epochs to reach the convergence. Since the 256 batch size was the boundary condition which performed the best, batch size 512 was also evaluated. But it does not train well and the validation accuracy remains stuck near 50%. hence batch size 256 is chosen for the final

evaluation.

Parameter size

Total parameters: 1,068,592, Trainable parameters: 1,068,336, Non-trainable parameters: 256 TODO for which configuration??

Final grid search

So all the best performing hyper parameter values are combined together for the final run.

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

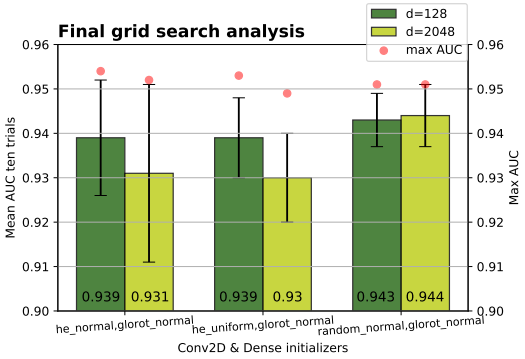


Figure 5.39: Final grid search analysis for VGG-Siamese network with contrastive loss.

Discussion The best result was obtained by RandomNormal and glorot normal as Conv and dense layer initializers respectively. Along with 128 and 2048 dense units values both resulting very close mean AUC values. So the best result is mean AUC (Ten trials) of 0.944 with std of 0.007 and highest AUC value in a single run as 0.95. The state of the art is AUC of 0.91, so the finding in this work hints at slight improvement.

Network	Mean AUC	Std	Max AUC	Total params
DenseNet two-channel	0.955	0.009	0.966	51430
DenseNet Siamese	0.921	0.016	0.95	16725485
Contrastive loss	0.944	0.007	0.951	3281840

Table 5.13: Comparative analysis on the AUC and total number of parameters in the best performing networks.

5.4 Comparative analysis

So the three network structures 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.

5.4.1 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. 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.

Discussion As seen in table 5.13 the total parameters for the DTC is much lower than the Siamese networks. For both Siamese networks, total parameter size is so

large because of the connection 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. If in DenseNet Siamese each branch has output size P parameters, then after merging or concatenation the total parameters becomes 2*P. If it is connected to the FC layer of output x (For example 2000) then the total parameters involved in that single computation step is 2*P*x. So in a single step, in Siamese networks the total parameters will increase by 2*x times P. That is why the total number of parameters for both Siamese are so much higher than DTC because it does not have this step.

Image index	Mean AUC(20)	Std(20)	Label
1000	0.909	0.082	1
1001	0.710	0.207	1
1002	0.617	0.188	1
1003	0.990	0.013	1
1004	0.996	0.003	1
1005	0.089	0.069	1
1006	0.645	0.166	1
1007	0.707	0.242	1
1008	0.989	0.016	1
1009	0.579	0.242	1
1010	0.032	0.026	0
1011	0.163	0.130	0
1012	0.025	0.062	0
1013	0.435	0.280	0
1014	0.507	0.254	0
1015	0.165	0.120	0
1016	0.249	0.170	0
1017	0.215	0.143	0
1018	0.146	0.117	0
1019	0.128	0.107	0

Table 5.14: Monte Carlo dropout analysis for DenseNet two-channel network, 20 trials of prediction has been done using MC dropout using a pre trained model. The original model have reported 0.965 AUC. The mean AUC and standard deviation of 20 iterations gives an idea about how sure the predictions of the network are. The less std the more sure it is. Also if the mean AUC is close to the label it's better

5.4.2 Monte Carlo Dropout analysis

MC Dropout analysis for Siamese networks

Dropout at inference time, does not work for Siamese though.

MC Dropout analysis for DenseNet two-channel

The evaluation presented in table 5.14 are for following images:

5.4.3 Real prediction analysis

5.4.4 Run time analysis

5.4.5 Total parameters analysis

5.4.6 Ensemble

It was observed that the performance of the DenseNet Siamese(DS) is good for non-matching pair predictions. DenseNet two-channel(DTC) is overall very good, so the idea was to see if their ensemble improves the overall prediction or not. So for the ensemble the output prediction of both the networks are **averaged** for each of the test data. The results are showing slight improvement over the individual prediction accuracy. For this test few of the previously trained models of DTC and DS are loaded in the memory, and their prediction on the test data is averaged. The ROC AUC calculated on the average prediction is found to be higher than the individual scores each time. The detailed evaluation result is displayed below.

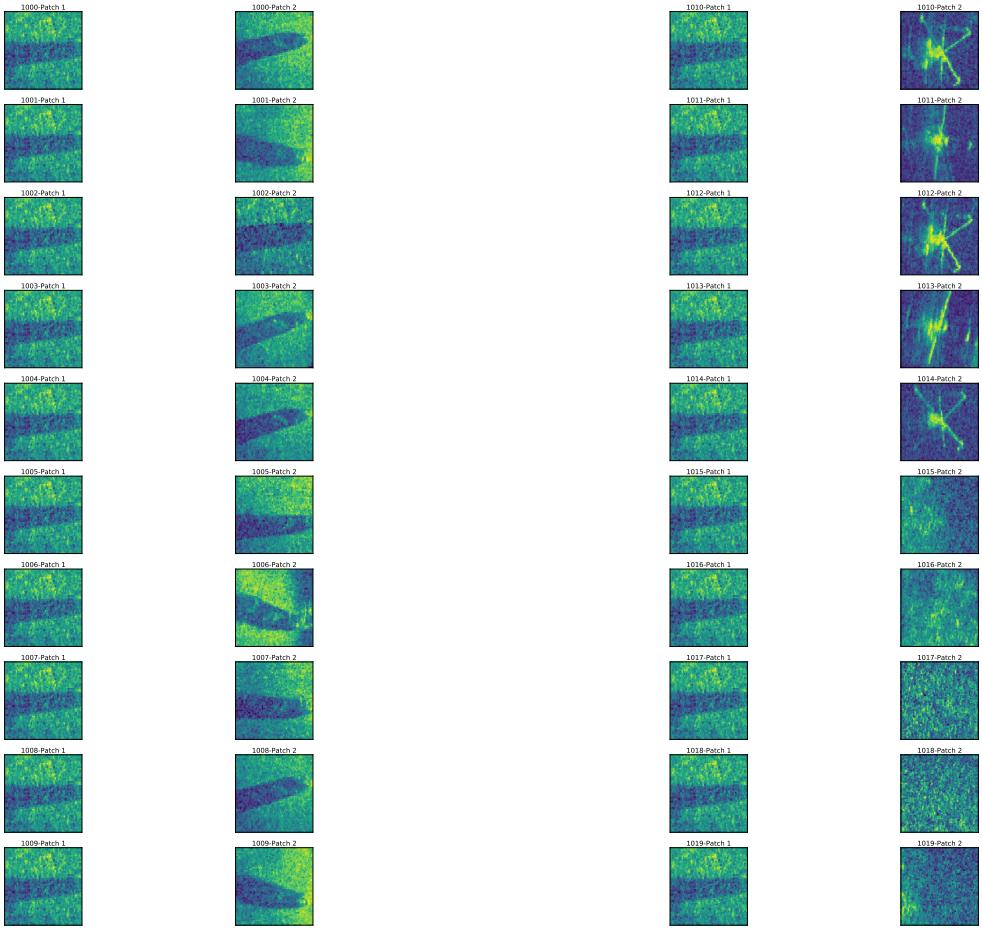


Figure 5.40: The images for the MC dropout test, not random but balanced match not match samples, will be changed later on after more analysis

6

Conclusions

6.1 Contributions

6.2 Lessons learned

6.3 Future work

References

[1] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems*, pages 737–744, 1994.

[2] Matthew Brown and David G Lowe. Automatic panoramic image stitching using invariant features. *International journal of computer vision*, 74(1):59–73, 2007.

[3] François Chollet et al. callbacks keras, December 2018. URL `https://keras.io/callbacks/`.

[4] François Chollet et al. Conv2D keras, December 2018. URL `https://keras.io/layers/convolutional/`.

[5] François Chollet et al. Binary cross entropy loss keras, December 2018. URL `https://keras.io/losses/#binary_crossentropy`.

[6] François Chollet et al. Initializers keras, November 2018. URL `https://keras.io/initializers/`.

[7] François Chollet et al. optimizers keras, November 2018. URL `https://keras.io/optimizers/`.

[8] François Chollet et al. Sigmoid keras, December 2018. URL `https://keras.io/activations/#sigmoid`.

[9] Thibault de Boissiere et al. Densenet implementation, December 2018. URL `https://github.com/tdeboissiere/DeepLearningImplementations/blob/master/DenseNet/densenet.py`.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[11] Simon Emberton, Lars Chittka, and Andrea Cavallaro. Underwater image and video dehazing with pure haze region segmentation. *Computer Vision and Image Understanding*, 168:145–156, 2018.

[12] John Hunter et al. (matplotlib team). Make a box and whisker plot, December 2018. URL `https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.boxplot`.

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

[14] The HDF Group. What is hdf5, December 2018. URL `https://support.hdfgroup.org/HDF5/whatishdf5.html`.

[15] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.

[16] Natalia Hurtós, Yvan Petillot, Joaquim Salvi, et al. Fourier-based registrations for two-dimensional forward-looking sonar image mosaicing. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5298–5305. Ieee, 2012.

[17] Natalia Hurtós, Narcís Palomeras, Sharad Nagappa, and Joaquim Salvi. Automatic detection of underwater chain links using a forward-looking sonar. In *OCEANS-Bergen, 2013 MTS/IEEE*, pages 1–7. IEEE, 2013.

[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[19] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[20] Juhwan Kim and Son-Cheol Yu. Convolutional neural network-based real-time rov detection using forward-looking sonar image. In *Autonomous Underwater Vehicles (AUV), 2016 IEEE/OES*, pages 396–400. IEEE, 2016.

[21] K Kim, N Neretti, and N Intrator. Mosaicing of acoustic camera images. *IEE Proceedings-Radar, Sonar and Navigation*, 152(4):263–270, 2005.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86 (11):2278–2324, 1998.

[23] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, page 20, 2015.

[24] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[25] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[26] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[27] Somshubra Majumdar(Github:titu1994) et al. Keras contrib implementation of Densenet, December 2018. URL `https://github.com/keras-team/keras-contrib/blob/master/keras_contrib/applications/densenet.py`.

[28] Nicholas Molton, Andrew J Davison, and Ian D Reid. Locally planar patch features for real-time structure from motion. In *Bmvc*, pages 1–10. Citeseer, 2004.

[29] S Negahdaripour, MD Aykin, and Shayanth Sinnarajah. Dynamic scene analysis and mosaicing of benthic habitats by fs sonar imaging-issues and complexities. In *Proc. OCEANS*, volume 2011, pages 1–7, 2011.

[30] Minh Tân Pham and Didier Guériot. Guided block-matching for sonar image registration using unsupervised kohonen neural networks. In *Oceans-San Diego, 2013*, pages 1–5. IEEE, 2013.

[31] Sebastian Ruder. Optimization for deep learning highlights in 2017, November 2018. URL `http://ruder.io/deep-learning-optimization-2017/index.html#understandinggeneralization`.

[32] scikit learn. Train test data split, December 2018. URL `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`.

[33] Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *null*, pages 519–528. IEEE, 2006.

[34] Tensorflow. Tensors, December 2018. URL `https://www.tensorflow.org/programmers_guide/tensors`.

[35] Matias Valdenegro-Toro. Objectness scoring and detection proposals in forward-looking sonar images with convolutional neural networks. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pages 209–219. Springer, 2016.

[36] Matias Valdenegro-Toro. Improving sonar image patch matching via deep learning. In *Mobile Robots (ECMR), 2017 European Conference on*, pages 1–6. IEEE, 2017.

[37] Peter Vandrish, Andrew Vardy, Dan Walker, and OA Dobre. Side-scan sonar image registration for auv navigation. In *Underwater Technology (UT), 2011 IEEE Symposium on and 2011 Workshop on Scientific Use of Submarine Cables and Related Technologies (SSC)*, pages 1–7. IEEE, 2011.

[38] Simon Wiesler and Hermann Ney. A convergence analysis of log-linear training. In *Advances in Neural Information Processing Systems*, pages 657–665, 2011.

[39] Wikipedia. Convolutional neural networks, November 2018. URL `https://en.wikipedia.org/wiki/Convolutional_neural_network`.

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

[41] Bangpeng Yao, Gary Bradski, and Li Fei-Fei. A codebook-free and annotation-free approach for fine-grained image categorization. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3466–3473. IEEE, 2012.

[42] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4353–4361, 2015.

[43] Jure Zbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research*, 17(1-32):2, 2016.