Siamese Network to image classification

Thomas Borsani
tborsani@unibz.it
Free University of Bozen-Bolzan

March 16, 2023

Abstract

Neural networks are a type of machine learning algorithm modelled after the structure and function of the human brain. One popular type of neural network is the Siamese Network, designed to compare two similar inputs and determine their similarity. Siamese networks are commonly used in face recognition, signature verification, and text similarity comparison tasks. This report provides an application for detecting if two satellite images show the same object.

Introduction

Neural networks are machine learning algorithms modelled after the structure and function of the human brain. One popular type of neural network is the Siamese Network, designed to compare two similar inputs and determine their similarity. Siamese networks are commonly used in face recognition, signature verification, and text similarity comparison tasks. This report provides an application to detect whether two satellite images show the same object.

A Siamese Network is a type of neural network architecture designed to compare two similar inputs and determine their similarity. It consists of two or more identical sub-networks that process each input independently and then merge their outputs to produce a final similarity score. During training, the weights of the sub-networks are updated in unison to effectively recognise features in the inputs that are relevant for determining similarity.

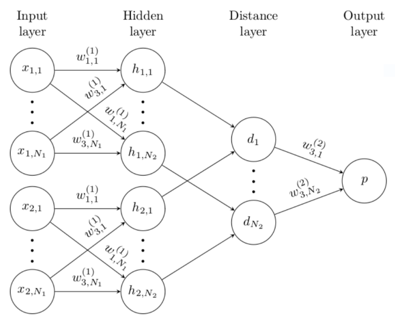


Figure 1: Siamese Network schema presented by Koch et.al [8]

Weight sharing ensures that if two inputs are similar, they will not be mapped to significantly different locations in feature space by the respective sub-networks [8]. The reason is that both sub-networks perform the same computations, resulting in identical feature representations. As a result, shared weights help to ensure that the Siamese Network accurately measures the similarity between two inputs.

This type of network was introduced by Bromley and LeCun in 1993 [5]. Their goal was to identify the fake signatures by checking if they were similar. This first paper proposes two identical Time Delay Neural Networks [9], but they do not suggest that the two networks share the weights.

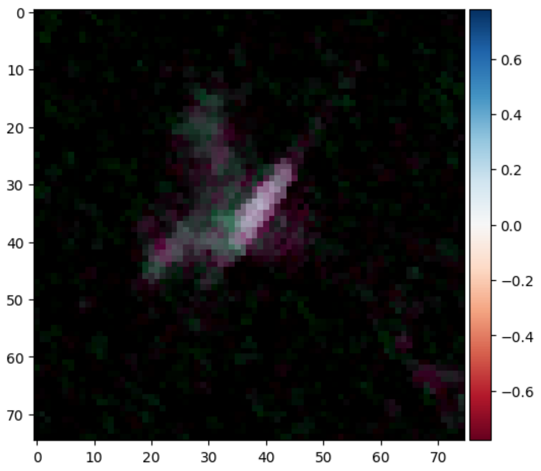
A more advanced Siamese Network architecture was proposed by LeCun et al. in 2006 [6]. In this paper, the objective was to verify the similarity between faces. Here the authors designed two convolutional networks with shared weights that converge in a unique, fully connected layer. The result of this architecture provides, as a result, the similarity between the two images as output. To train the network, the authors defined a Contrastive Loss Function. This loss function comprises two separate loss functions, one for the real image and one for the fake image. The Contrastive Loss Function enables the network to learn the similarities and differences between the two inputs and accurately measure their similarity.

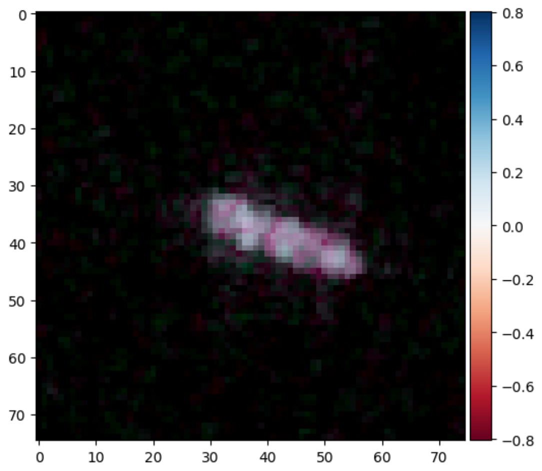
In this project, we design a siamese network architecture with two convolutional networks that converge in one fully connected network to determine if two satellite photos depict the same element.

1 Methodology

1.1 Dataset

The dataset used in this project is freely available on Kaggle [1]. It consists of 4113 satellite photos of icebergs or ships. These are RGB images with a resolution of 75x75. Each image has a label that defines its class.


Iceberg


Ship

The dataset is split into three parts: a training set (64%), a validation set (16%), and a test set (20%). We have decided to work with normalised images.

To use the dataset in a Siamese Network, we must create a set of image pairs to compare. We have decided to compare each image to 5 different images, duplicated from the original dataset, ensuring that we do not compare the same image. Additionally, it is necessary to create a new target variable which indicates whether two images have the same label (0) or not (1). These datasets of pairs will then be fed into the Siamese Network to determine their similarity.

1.2 Data Augmentation

To improve the number of observations in the dataset and the ability of the model to generalise better and reduce overfitting, we use data augmentation. We use three data augmentation strategies: HorizontalFlip, VerticalFlip, and Grayscale. HorizontlFlip is a technique for flipping the image horizontally along its vertical axis, creating a new and different image from the original. While, VerticalFlip is a technique for flipping the image vertically along its horizontal axis, creating a new and different image from the original. Grayscale is a technique that transforms an image colour into a scale of grey.

More in detail, we use a random data augmentation with a θ of 0.7. This means that the augmentations are applied randomly with a probability of θ. We use data augmentation only in the training set, which is applied to both elements of the pairs. By visualizing the first five pairs, we obtain figure 2

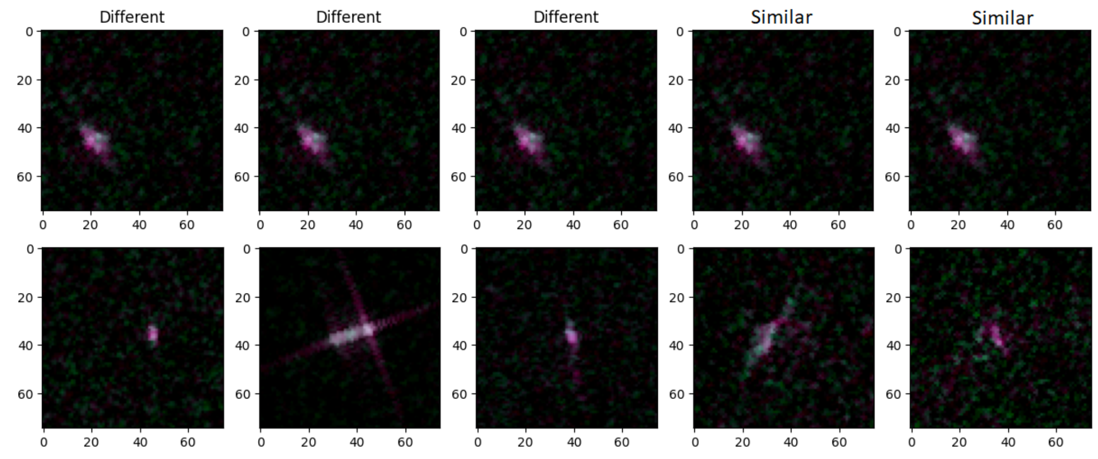


Figure 2: five pairs with data augmentation at random

1.3 Network architecture

As previously mentioned, we have designed a Siamese Network. The network design is based on a Kaggle notebook [2]. The Siamese Network comprises two identical convolutional networks, each of which has three sequential blocks. The first two blocks consisting of a convolutional layer followed by a ReLU activation function and a max pooling layer. The convolutional layer uses a 3x3 kernel with a stride and padding of 1, while the max pooling layer has a 2x2 kernel and a stride of 2.

The final block is a convolutional layer with a ReLU activation function, whose units are flattened into a single vector. Then we use one from a fully connected layer followed by a sigmoid activation function. This part is trained on the two images of the comparison pair. The two results are then joined using a distance L1. The resulting metric is given to a fully connected layer that returns a single output unit. The final result is a tensor of size Bx1, with B being the batch size.

```
SiameseNetwork(
  (conv): Sequential(
    (0): Conv2d(3, 5, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(5, 5, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(5, 7, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU()
    (8): Flatten(start_dim=1, end_dim=-1)
  )
  (fc1): Sequential(
    (0): Linear(in_features=2268, out_features=18, bias=True)
    (1): Sigmoid()
  )
  (out): Linear(in_features=18, out_features=1, bias=True)
)
```

1.4 Weight initialisation

The weight initialisation strategy combines the weight initialisation proposed by Koch et al. [8] and the choice to use ReLu as an activation function. In fact, it is documented that the performance of a model that uses the ReLu activation function can be improved by initialising the weight from a normal distribution with variance $\frac{2}{D_in}$ defining D_in the number of input channels in the layer [7].

Therefore we initialize the weights for the convolutional layers drawing from a normal distribution with mean zero and variance $\frac{2}{D_in}$. At the same time, the biases of the convolutional layers are drawn from a normal distribution with a mean of 0.5 and variance $\frac{2}{D_in}$. For the fully connected layer, we draw the weights from a normal distribution with a mean equal to zero and a variance of $\frac{2}{D_in}$. The biases are extracted in the same way as for the convolutional layer.

1.5 Loss function

Binary cross entropy is a commonly used loss function for binary classification problems. It measures the difference between the predicted probability and the actual label, giving a scalar value that can be used to optimise the network's parameters, as shown in the formula 1. In this case, where the target variable is binary (similar or not similar), and the neural network result is a value that represents the similarity of the two input images, the binary cross entropy loss function could be seen as an appropriate choice.

$$L(y_i, \hat{y_i}) = -y_i log(\hat{y_i}) - (1 - y_i) log(1 - \hat{y_i}) \tag{1}$$

1.6 Optimization

For optimisation, we use the ADAM optimiser with a learning rate of 0.01 and a weight decay of 2.5e-4. We also use a learning rate scheduler to improve the optimisation process further. In particular, we use an exponential scheduler with a gamma of 0.9 per epoch, as in equation2

$$\eta_t = \eta \alpha^t \tag{2}$$

1.7 Training

We divide the data sets into batches of 64 images and train the network for 30 epochs.

1.8 Evaluation strategy

To evaluate the model's performance, we use different measures. The primary measure used is the value of the loss function to monitoring if the network while training. We use WandB [4] to visualise model metrics stream live. We also implement other measures to control the model's performance; we compute the cosine similarity between the predicted output and the target. We use the cosine similarity to ensure a distance measure contained in $[0, 1]$. Another advantage of using cosine similarity is that it is independent of the magnitude of the vectors, so it only considers the direction of the vectors. Therefore, we define a threshold of 0.5 to determine if the network results are similar or dissimilar.

With the defined threshold, we can then calculate the confusion matrix, which includes the True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). TP represents the number of observations correctly classified as similar, TN represents the number of observations correctly classified as dissimilar, FP represents the number of false positives (incorrectly classified as similar), and FN represents the number of false negatives (incorrectly classified as dissimilar). We then rely on the Accuracy and Jaccard index as here defined

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FN + FP)}$$
$$Jaccard = \frac{TP}{(TP + FN + FP)}$$

1.9 Experiment strategy

To reach the hyperparameters and optimisation strategy results discussed above, we start from the only standard we found on Kaggle [2] for the data set in exams. Therefore the first step was to reproduce the result proposed by the author and assume that as a baseline. Consequently, our baseline involves no weight initialisation, a learning rate scheduler, or data augmentation, training the network over 20 epochs with a learning rate of 0.001.

We conducted a manual search to determine the best hyperparameters and optimise the network's performance. As a guideline, we make use of the Tuning playbook published by Godbole and colleagues[3].
First, we identify the best loss function among the binary cross entropy, the Contrastive Loss function proposed by LeCun et al. in 2006 [6] and the Loss function implemented by Koch et al. [8].

As a second step, we focus on a learning rate more suitable for our problem; this also involves using an exponential scheduler to improve the neural network by reducing the learning rate over time as the network approaches convergence. We also increased the number of epochs until the training and validation loss reached a better optimal solution.

In the third step, we focus on improving the ability of the network to generalize the results. These steps include using dropout layers, batch normalisation and data augmentation. These techniques prevent overfitting by reducing the dependence of the neural network on specific features and adding variability to the training data. This results in a neural network that can generalize better and produce accurate results on unseen data.

In the fourth step, we focus on improving the neural network by changing the weight initialisation. Weight initialisation is a relevant aspect as it determines the starting point of the optimisation process. Poorly initialised weights can lead to vanishing or exploding gradients, slowing down or preventing convergence to an optimal solution.

The final step to improve the neural network's performance is using an ensemble approach by combining the predictions of two models with different weight initialisations. These techniques can lead to a more robust and accurate model, as the ensemble model will be less prone to overfitting and benefit from both models' strengths.

2 Results

The performance obtained by what we define as the baseline is good on the validation set but needs to be more generalisable to the test set. As shown in figure 3, the network is overfitting.

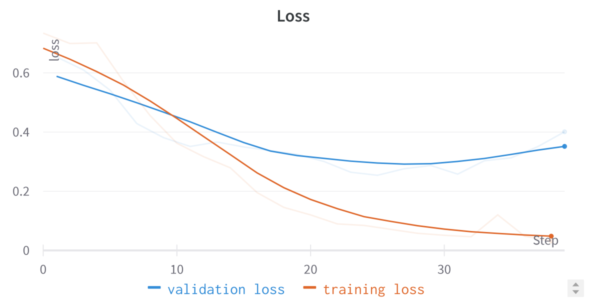


Figure 3: Training and validation loss of the baseline neural network averaged with moving window

After exploring diverse configurations, the best solution is achieved by combining all the hyperparameters and techniques discussed in the methodology section (Section 1). As shown in Figure 4, the final model has a decreasing and converging loss for both the training and validation sets, reaching an optimal solution.

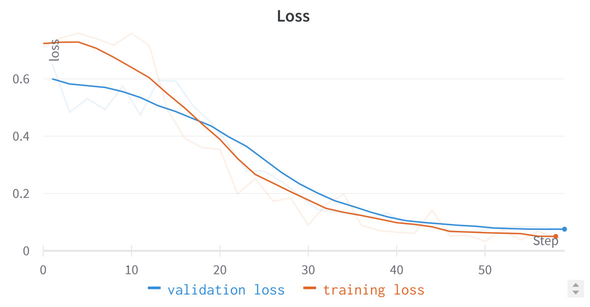


Figure 4: Training and validation loss of the final neural network using He weight initialisation on the right. Averaged with moving window

The final model with the He initialization leads to an accuracy of 96.14% on the test set and a Jaccard at 92.76% over the previous result in the baseline of 88.46% and Jaccard at 81.09%.

We then test our final model on a classification task. For this proposal, we consider only the image of a particular class, ships, in one dataset and compare it with randomly selected other images from the test set, ensuring that we do not compare the same image. Because we are worried about reducing the test set excessively, we compare each ship image with 15 other photos. Our Siamese network can classify correctly 5938 out of 6195 images that, lead to an accuracy of 95.85% and Jaccard at 92.39%

In the final phase of our experiment, we aim to comprehend what the network extracts from the original images to determine their similarity. To do this, we utilise a method known as saliency map visualisation. This technique offers insight into which parts of the pictures impact the network's similarity prediction.

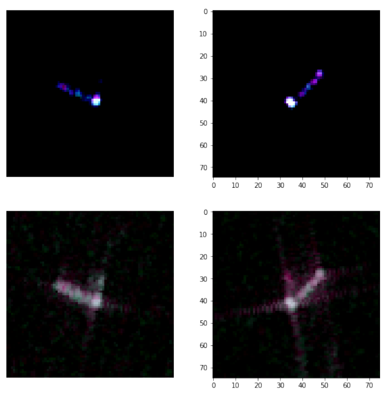


Figure 5: Visualisation with patch cam of our Siamese network identifying if the image on the right is a ship. In this example, the prediction is correct.

3 Discussion

This study demonstrates the process of enhancing the performance of a Siamese neural network using a binary cross-entropy loss function. The approach involved identifying and implementing improved hyperparameters, regularisation techniques, and optimisation methods to find the minimum of the loss function effectively. It is possible to improve the reported accuracy measure further by determining an optimal threshold; a common practice is identifying the threshold as the point that maximises the F1 score or the AUC under ROC.

Notice that we use cosine similarity to determine whether the prediction is correct. We decided to use this measure because the network output reports the similarity between two images and not the probability that the photos belong to a particular class.

Following the same thought, we also consider that the binary cross entropy could not be politely suited for this task. We, therefore, try to change the loss function using the Contrastive Loss function[6]. This loss function measures the difference between two image pairs and tries to minimise the difference between them. The aim is to have the representations of similar samples close together while the representations of dissimilar samples are far apart in the learned feature space. Even though it could be more appropriate as a loss function, we demonstrate the reliable performance of using a binary cross-entropy loss function.

The use of the patch cam confirms the robustness of our results. Indeed, we can see that the network recognises the shape of a ship in both images and thus attributes the desirable similarity between the objects. Consequently, using a binary cross entropy leads to a good solution.

Introducing a dropout layer in the neural network does not lead to a suitable solution for the low reduced dimension of our network. More promising was the use of an ensemble network combining the result of the network with two different weight initialisation. But the weight initialisation proposed by Koch et al. [8] led the training loss to a not converging solution, as seen in Figure 6. Therefore when we tested to combine the networks with the two different weight initialisation, the result did not reach the desired outcome.

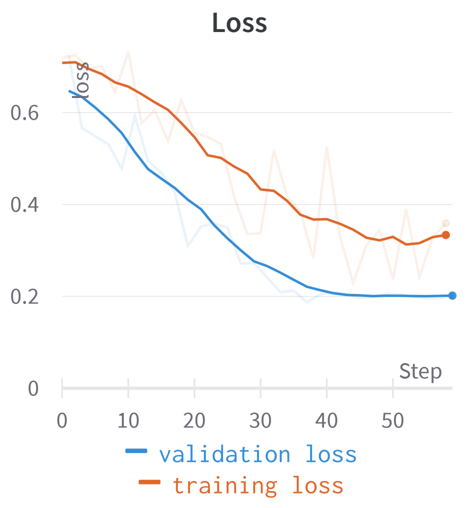


Figure 6: Training and validation loss of the final neural network using Koch weight initialisation on the right. Averaged with moving window

4 Conclusion

This work shows how results can be improved from a shallow neural network by identifying better hyperparameters, regularisation techniques and optimisers to navigate the space of loss functions with gradient descent more efficiently. Better navigation in the space of loss functions means a better optimal solution that leads to better results.

References

[1] Kaggle. https://www.kaggle.com/datasets/saurabhbagchi/ship-and-iceberg-images.

[2] Kaggle. https://www.kaggle.com/code/saurabhbagchi/ship-and-iceberg-dataset-for-starters.

[3] Tuning playbook. https://github.com/google-research/tuning_playbook/blob/main/README.md.

[4] wandb. https://wandb.ai/site/experiment-tracking.

[5] Jane Bromley, Isabelle Guyon, Yann Lecun, Eduard Säckinger, and Roopak Shah. Signature verification using a siamese time delay neural network. volume 7, pages 737–744, 01 1993.

[6] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546 vol. 1, 2005.

[7] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[8] Gregory R. Koch. Siamese neural networks for one-shot image recognition. 2015.

[9] I. Ramadass Subramanian, P. Albrecht, Yann LeCun, John S. Denker, and Wayne E. Hubbard. A time delay neural network character recognizer for a touch terminal. 1990.