Removing parasitic elements from Quantum Optical Coherence Tomography data with Convolutional Neural Networks

Krzysztof A. Maliszewski [1] **Sylwia M. Kolenderska** [2] **Varvara Vetrova** [1]

Abstract

Quantum Optical Coherence Tomography (Q-OCT) is a non-contact and non-invasive light-based imaging method which is gaining attention due to its increased image resolution and quality. The biggest, yet unresolved, disadvantage of Q-OCT is artefacts, additional elements cluttering the images, and leading to a loss of the structural information in the obtained images. In our work, Convolutional Neural Network (CNN) is applied to remove artefacts from Quantum Optical Coherence Tomography (Q-OCT) images. In our approach, we train our model with computer-generated data instead of experimental images. The preliminary results show that such an approach is successful in retrieving artefact-free structural information, even for multilayer objects, for which this information is lost due to the number of induced artefacts. The limitations and challenges associated with our approach are identified and discussed.

1. Introduction

Optical Coherence Tomography (OCT) is a non-contact and non-invasive light-based technique providing images of the inside of semi-transparent objects, such as eyes or skin (De Boer et al., 2017). Quantum OCT (Q-OCT) (Kolenderska et al., 2020a) uses the quantum nature of light to provide many advantages over standard classical OCT imaging: two-fold resolution increase and immunity to resolution-degrading chromatic dispersion. Unfortunately, the same quantum effects that are responsible for Q-OCT's extraordinary features give rise to its one huge drawback: parasitic elements called artefacts. Artefacts are additional peaks on images that do not relate to the structure of the object and effectively, lead to the scrambling of the image. There are already both hardware (Graciano et al., 2019) and software (Kolenderska & Szkulmowski, 2021) schemes to reduce artefacts and, in some well-defined cases, suppress them completely, but they pose some requirements on the experimental system, which in the case of very thin-layered objects are impossible to meet.

(Maliszewski & Kolenderska, 2021) showed for the first time the possibility of removing Q-OCT artefacts with machine learning, even for objects with thin layers. They use computer-generated data during training, and their models return just 256-pixel-long outputs. Such results provide a very limited application in experimental situations which usually expect at least 1024-pixel-long outputs. Lastly, their outputs contain idealised A-scans that are reflectivity values at the object interfaces' positions. This approach is not a natural representation of the OCT data and, as suggested in the paper, introduces problems with detecting interfaces in the presence of noise.

We improve upon the approach and present a method with longer outputs that, in principle, allow imaging of deeper layers inside samples and are preferred in experimental A-scans. The ground truth signals better represent the experimental signals, making them easily understandable to professionals in the OCT field. This approach raises new challenges that are identified and discussed in this paper.

2. FFT stacks and artefacts

The core of our approach is a signal called an FFT stack whose elements exhibit behaviour that can be uniquely related to the object structure.

FFT stack (Figure 1c) is a stack of A-scans which are generated by Fourier transforming the diagonals (Figure 1b) of a joint spectrum (Figure 1a), (see (Kolenderska et al., 2020b) for more detailed information on what a joint spectrum is). The FFT stack contains two types of elements: vertical solid lines representing the structural peaks and vertical intensity-oscillating lines, which are artefacts. For a three-interface object as presented in Figure 1, there are three solid lines which correspond to the three interfaces of the object, and,

[1]Department of Mathematics and Statistics, University of Canterbury, Christchurch, New Zealand [2]School of Physical and Chemical Sciences, University of Canterbury, Christchurch, New Zealand. Correspondence to: Krzysztof Adrian Maliszewski <kma338@uclive.ac.nz>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

in addition, there are six oscillating artefact elements. As a rule, there are always two artefacts for every pair of interfaces in an FFT stack. This leads to complete scrambling of the structural information, as shown in Figure 1d where one A-scan from the FFT stack (in dotted grey) is plotted together with its artefact-free equivalent (in blue).

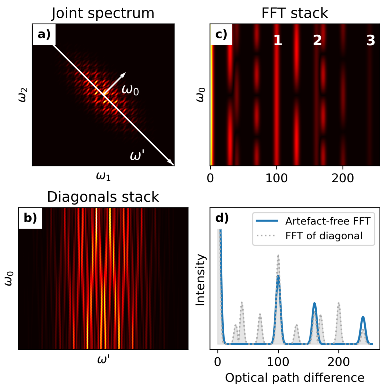

Figure 1. (a) A Q-OCT joint spectrum of a three-interface object, ω_1 and ω_2 are optical frequencies of photons in a photon pair whose quantum interference in Q-OCT leads to the creation of a joint spectrum. (b) A stack of diagonals is extracted symmetrically from around the main diagonal of the joint spectrum. (c) An FFT stack created by Fourier transforming each row of a stack of diagonals; structural peaks are represented by solid lines (1, 2 and 3) and artefacts by oscillating lines. (d) One row from the FFT stack shows an A-scan cluttered with artefacts (in gray); the true structure of the object in the form of an artefact-free A-scan is presented in blue.

In the case of simple object structures, it is generally easy to determine where the structure and artefacts are because all elements tend to be separated (Figure 2 a,c). However, even for simple object structures, a situation may happen where an artefact and an interface peak overlap (Figure 2 b,d) leading to their mutual interference. In structurally complicated objects (Figure 2 e,f), artefacts overlap with interfaces as well as with other artefacts making the A-scan indecipherable.

Each FFT stack contains a pattern that is uniquely related to an object with a specific internal structure. We cast this problem as a supervised machine learning task, where the goal is to predict A-scans based on FFT stacks.

3. Data preparation

Our dataset is entirely computer-generated according to the algorithm introduced in (Kolenderska & Kolenderski, 2021). Obtaining a huge number of Q-OCT images experimentally is time-consuming and there are no publicly-available online Q-OCT datasets. Using experimental images would also present extreme challenges in determining the correct geometrical distances inside an object and the object's optical properties, which are crucial for training. Due to the hardware limitation of our machine and the sheer size of datasets, we generate our data on the fly during training.

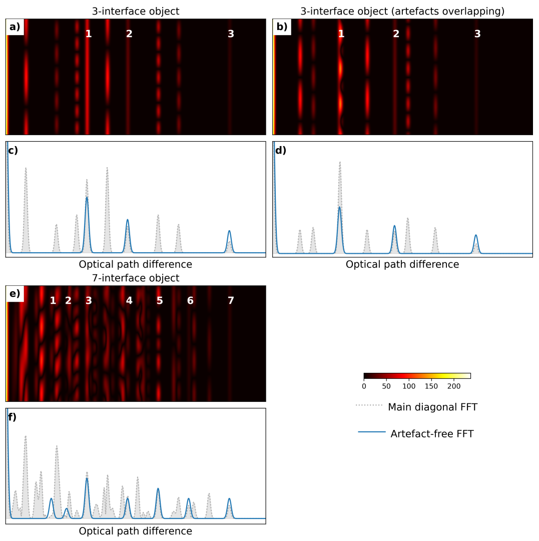

Figure 2. (a,b,e) FFT stacks and (c,d,f) single rows from the FFT stack (in gray) together with their artefact-free equivalents (in blue). (a, c) A three-interface object for which there is a clear distinction between artefacts and structural peaks. (b,d) A three-interface object for which an artefact overlaps with the first interface and causes an oscillation in place of the solid line. (e,f) Multi-interface object for which the structural information about the object is lost due to the high number of overlapping artefacts.

Our input FFT stacks have a dimension of 50 by 1024 and single precision. We generate objects that consist of a random number of interfaces, up to 12, located at random positions up to the 512th pixel. We Fourier transform 50 diagonals of the joint spectrum and zero-pad them to be 2048-elements-long. We take only half of each of the obtained Fourier transforms (for real-valued signals, the same information is repeated twice in a Fourier transform) and normalize them.

Our outputs are one-dimensional 1024-pixel-long sparse vectors containing values in the range [0,1]. They are artefact-free A-scans calculated by Fourier transforming a computer-generated, resolution-doubled classical OCT spectrum.

The generated data represents spectra with 840 nm central wavelength, and with a total spectral bandwidth of 160 nm and a Gaussian profile. We do not incorporate noise into our datasets.

4. Dataset size

First, we optimised the amount of data needed for training to obtain satisfactory results in a reasonable time. As the initial architecture of our models, we used the VGG16 model (Simonyan & Zisserman, 2014). VGG is good at extracting low-level image features, such as textures, lines, and points, whereas other modern architectures, such as ResNet (He et al., 2016) or Xception (Chollet, 2017), provide high-level features that are not relevant to our problem. We modified the architecture according to (Maliszewski & Kolenderska, 2021) proposal by adding batch normalization (Ioffe & Szegedy, 2015) after each convolutional layer. We further altered that model by changing the output shape to 1024 nodes. We trained models from scratch with learning rate 0.0001, batch size 16, optimiser Adam (Kingma & Ba, 2014), with 16,000, 32,000, 64,000, 128,000, and 256,000 training examples. We measured the performance using the mean squared error (MSE) metric. During training, we used the V100 GPU with 12GB of memory.

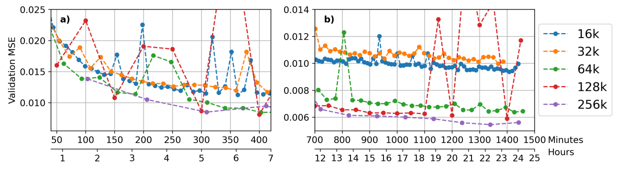

Figure 3. Average training MSE as a function of time of training. Five models were trained with datasets containing a different number of samples. All models were trained for around 24 hours.

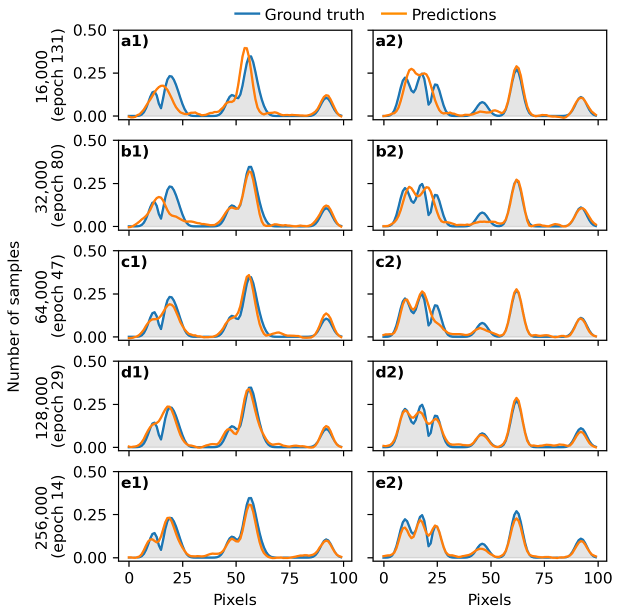

Figure 4. Predictions, orange lines, and ground truths, blue lines, for models trained with datasets with a different number of samples. Each row presents a different dataset size. Each column shows a different object to predict.

We trained models with each dataset for around 24 hours. Within this time, the model trained with a dataset containing 16,000 samples completed 131 epochs, 32,000 - 80 epochs, 64,000 - 47 epochs, 128,000 - 29 epochs, and 256,000 - 14 epochs. fig:hrs presents how validation MSE changed during the trainings. In Figure 3, we observe loss fluctuations especially visible for the 128,000 model. We address this issue in following sections 5 and 6. In Figure 3a, we can see that initially the networks trained on smaller datasets were closely following models trained on the bigger datasets. However, at the end of the test, the validation loss of the 16,000 and 32,000 models was twice that of the models trained with the larger datasets. Additionally, we noticed that relatively low validation MSE values for models trained with smaller datasets do not necessarily translate into overall good predictions. Figure 4 illustrates this phenomenon by comparing the prediction results for two randomly selected objects from the test dataset (marked with 1,2 placed after the letters) using various sizes of the training sets. Each dataset size is presented in a separate row (marked with a-e). Figure 4 a,b,c show predictions, in orange, that poorly approximate the ground truth, in blue. In Figure 4d, we could observe model's results get closer to the expected values. Model trained with the largest dataset, 256,000 samples, provides us with the best predictions (Figure 4e). Models trained on smaller datasets give worse predictions due to the fact the datasets do not have enough data samples that would represent a wide range of variability of the structural information of objects to ensure that a model can "recognize" more distinctive examples.

After taking all of it into account, we decided to train our models with datasets containing around 256,000 training samples. In addition to 256,000 training samples, we generate 15,000 validation samples and 5,000 test samples. All datasets comprised of proportionally equal number of samples for the given number of interfaces within the objects.

5. Zero-convergence problem

Initially, we decided to look at our problem as a classification problem. The models used to find the optimal dataset size used Cosine Similarity as a loss function and sigmoid as an activation function in the output layer. The combination of these functions creates two major problems for a model trained with our 1024-pixel-long outputs. Firstly, in many cases, a model is unable to separate closely located structural interfaces, even for larger datasets, as shown in Figure 4 a2,b2. Secondly, the model's loss converges very fast to values close to zero, and its performance stops improving, although predictions are far from the ground truths (Figure 5 a,b,c).

We modified our model and applied binary-crossentropy as a loss function and sigmoid as an activation function in the output layer. Such a combination is widely used in multi-label and multi-classification problems. In contrast to the classification problem where output values are either 0 or 1, our outputs contain continuous values in the range [0,1]. Due to that fact and because our outputs are 1024-pixel-long

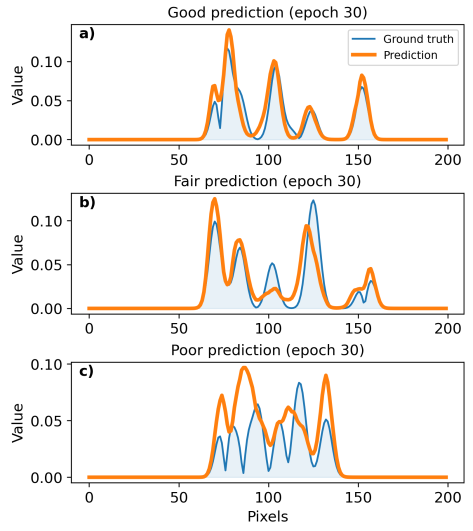

Figure 5. (a,b,c) Ground truth (blue line) and predictions (orange lines) corresponding to each test data sample after the network was trained for 30 epochs. Plots were trimmed from 1024 pixels to 200 pixels to zoom into the relevant part of the test data.

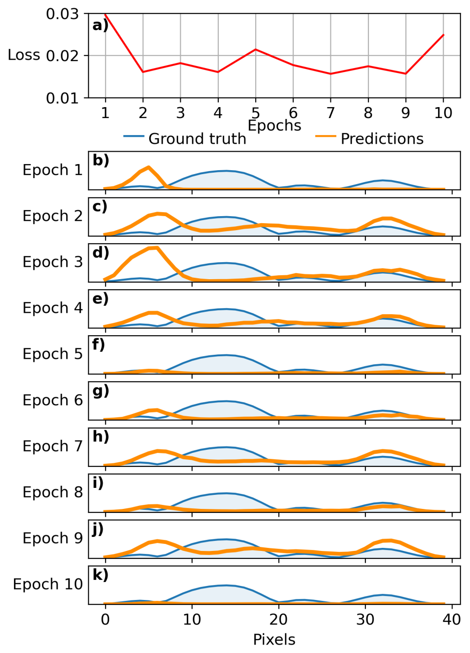

Figure 6. (a) Example of loss for the test data sample in the consecutive epochs during training. (b-k) Ground truth (blue) and predictions (orange) of the test data sample in the consecutive epochs.

sparse vectors, we observed a zero-convergence problem which we show in Figure 6.

Figure 6a presents the loss value corresponding to the test data sample in consecutive epochs. Figure 6b-k show predictions (in orange) of the test data sample in these epochs together with the ground truth (in blue). At the beginning of training, epochs 1-4, the model brings predictions closer to the sparse ground truth as shown in Figure 6a by changing its weights and consequently minimising its loss. In epoch 5, the prediction is very close to zero and very far from the ground truth. The loss becomes much higher than in the previous epochs and that causes the model to readjust the weights and to come closer to the ground truth again in epoch 6. In consecutive epochs, the process repeats.

Section 7 explains in detail why the zero-convergence problem occurs. To mitigate the zero-convergence problem, we switched to solving a regression problem.

6. Model instability

We applied the mean absolute error (MAE) as a loss function and used a linear function as an activation function in the output layer. Figure 7a presents MAE loss during 40-epoch-long training using a logarithmic scale. The red dots in the plot represent the epochs for which the predictions are shown in the graphs Figure 7b-j. The model loss until epoch 13, Figure 7b, decreases. In epochs 14 and 15, Figure 7c,d, the model becomes unstable, its predictions are far from the ground truth, and the loss increases dramatically. Then, training is carried out normally between epochs 16 and 33, Figure 7e,f. Epochs 34 and 36, Figure 7g,i, show another instability, but the model returns to normal in the following epoch 37, Figure 7j. Figure 7k shows the training loss (green line) and the validation loss (red line) in further training of the model. Despite the training loss gradually decreasing, instabilities occasionally occurred up to epoch 200 after which the model got completely destabilised and the validation loss shot up.

We partially solved the instability problem by adding a normalization layer before the output layer. This solution kept the values within the range [0,1].

7. Mean Absolute Error Thresholded and Goodness-of-Fit Thresholded

As we can see from Figure 7c, the issue of zero-convergence persists also in the regression problem. We posit that it exists because of the sparsity of ground-truth signals for the reasons we described above in Section 5. Our problem is the loss function, MAE. It does not account for the difference between near-zero values and values well above zero and divides absolute error by the length of the vector. That causes MAE to return much smaller errors than it should and points a model in the wrong direction. To finally solve the zero-convergence problem, we introduce a new loss function, which we call the Mean Absolute Error Thresholded (MAET):

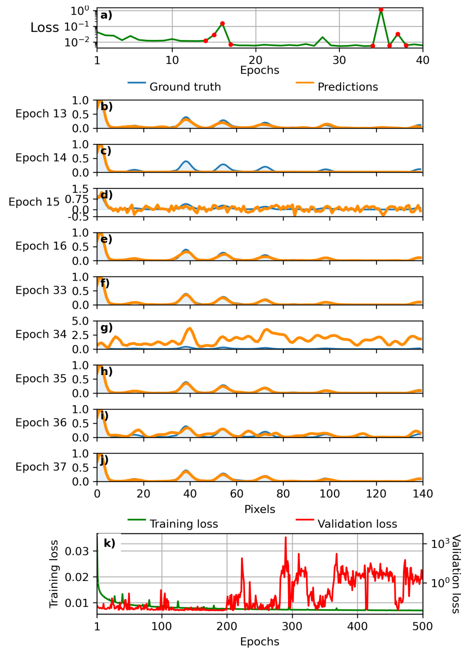

Figure 7. (a) MAE loss of the test data sample during 40 epochs of training. Y-axis is in the logarithmic scale. Green dots represent the epochs for which we calculate prediction on plots b-j. (b-j) Ground truth (blue line) and predictions of the test data sample (orange line) for particular epochs. (k) Training loss (green line) and validation loss (red line) of the model trained for 500 epochs. Y-axis is in the logarithmic scale.

$$MAET = \frac{\alpha}{N} \sum_{i=1, y_i > th}^{N} |y_i - \hat{y}_i| + \frac{\beta}{M} \sum_{i=1, y_i <= th}^{M} |y_i - \hat{y}_i|$$

where y and $\hat{y}$ are ground truth and prediction vectors, N and M are respectively, the number of pixels with a value above the threshold and under or equal to the threshold, with th being the threshold and α and β being coefficients that need to be specified.

MAET separately calculates the mean absolute error of ground-truth values above and under a given threshold th. It then multiplies them by α and β coefficients respectively and adds them to obtain a final loss value. We used Optuna (Akiba et al., 2019) to find optimal values of α, β and threshold hyperparameters. During the optimization process, we used MAET as a loss function and MAE as a metric upon which we decided the optimal values of hyperparameters. The optimal values were α=1., β=0.66 and th=0.005.

We created complementary metrics to MAET to calculate the goodness of fit of our predictions. We called them Goodness-of-Fit Above Thresholded (GoFAT), Goodness-of-Fit Under Thresholded (GoFUT) and Goodness-of-Fit Thresholded. All metrics are presented in Algorithms 1-3.

GoFT calculates the percentage of the prediction values, $\hat{y}$, that are within the maximum distance ($distA$ and $distU$) from the ground truth (y) for the ground-truth values above and below the given threshold th. GoFAT calculates that percentage only for values that are above the threshold, and GoFUT only for the ones below the threshold. We chose $distA$ equal to 0.01 and $distU$ equal to 0.005 as the acceptable distances.

Algorithm 1 Goodness-of-Fit Thresholded (GoFT)

```
**Input:** ground truth vector $\mathbf{Y}$ with length $N$, prediction vector $\hat{\mathbf{Y}}$ with length $N$, threshold $thr$=0.005, max deviation from $Y$ above the threshold $distA$=0.01, max distance from $\mathbf{Y}$ under the threshold $distU$=0.005
**d** = |**Y** - $\hat{\mathbf{Y}}$|
**for** $i \leftarrow$ 1 to N **do**
    $\mathbf{withinDistA}_i = \begin{cases} 1 & \text{if } d_i < distA \text{ and } y_i > thr \\ 0 & \text{otherwise} \end{cases}$
    $\mathbf{withinDistU}_i = \begin{cases} 1 & \text{if } d_i < distU \text{ and } y_i \leq thr \\ 0 & \text{otherwise} \end{cases}$
**end for**
**GoFT** = $\frac{1}{N}(\sum \mathbf{withinDistA} + \sum \mathbf{withinDistU})$
```

Algorithm 2 Goodness-of-Fit Above Thresholded (GoFAT)

```
**Input:** ground truth vector $\mathbf{Y}$ with length $N$, prediction vector $\hat{\mathbf{Y}}$ with length $N$, threshold $thr$=0.005, max deviation from $Y$ above the threshold $distA$=0.01
**d** = |**Y** - $\hat{\mathbf{Y}}$|
**for** $i \leftarrow$ 1 to N **do**
    $\mathbf{withinDistA}_i = \begin{cases} 1 & \text{if } d_i < distA \text{ and } y_i > thr \\ 0 & \text{otherwise} \end{cases}$
**end for**
**GoFAT** = $\frac{1}{N} \sum \mathbf{withinDistA}$
```

Algorithm 3 Goodness-of-Fit Under Thresholded (GoFUT)

```
**Input:** ground truth vector $\mathbf{Y}$ with length $N$, prediction vector $\hat{\mathbf{Y}}$ with length $N$, threshold $thr$=0.005, max deviation from $Y$ under the threshold $distU$=0.01
**d** = |**Y** - $\hat{\mathbf{Y}}$|
**for** $i \leftarrow$ 1 to N **do**
    $\mathbf{withinDistU}_i = \begin{cases} 1 & \text{if } d_i < distU \text{ and } y_i \leq thr \\ 0 & \text{otherwise} \end{cases}$
**end for**
**GoFUT** = $\frac{1}{N} \sum \mathbf{withinDistU}$
```

8. Architecture

We used a modified VGG16 architecture as a base for our model. Table 1 presents the set of hyper-parameters that were tuned in our model with Optuna. The best performing model contains batch normalisation layers after each convolutional layer, a max-pooling layer, and a single fully connected layer with 14336 units and a dropout rate (Srivastava et al., 2014) of 0.1. As discussed in Section 6, we also add a layer normalization before the output. Figure 8 presents the final architecture of our model.

Table 1. Hyper-parameters of the model architecture optimised with Optuna and their optimal values.
[A..B, C] - A and B are values of the limits of the range, and C is the step size.

Hyper-parameter	Values	Optimal
Batch normalization layer after a convolutional layer	True, False	True
Type of pooling	max, avg	max
Number of fully connected layers	[1..4, 1]	1
Number of units in each fully connected layer	[1024..16384, 1024]	14336
Dropout rates	[0.1..0.5, 0.05]	0.1
Learning rate	[1e-6, 1e-1]	2e-4
Optimizer	[Adam, Nadam, SGD, RMSProp]	Adam

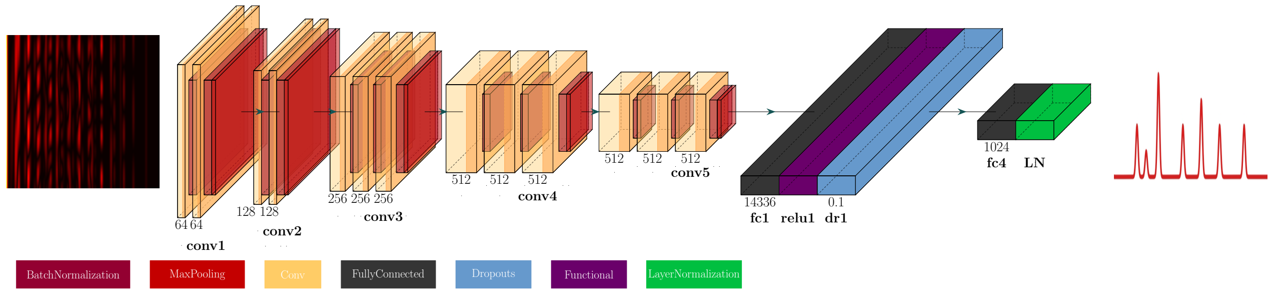

Figure 8. The architecture of the model used for the removal of artefacts from Q-OCT signals. The plot was created with PlotNeuralNet software (Iqbal, 2018).

9. Training

We have created a model using Python. We used an open-source software library TensorFlow (Abadi et al., 2016) v2.5.0. The training parameters of the models were optimised using Optuna (optimizer Adam, learning rate 2e-4). As presented in Section 7, we use MAET loss function with parameters $\alpha = 1.$, β =0.66, th =0.005. We used default values for the goodness-of-fit metrics. We set the batch size to 16 and trained our model from scratch for 60 epochs. Each epoch took about 1.5 hours where more than half of that time is spent on on-fly data generation.

10. Results

10.1. Performance

Figure 9 shows GoF values during training (in blue) and validation (in orange) together with the MAET loss values during training (in green) and validation (in red). The performance of the model gradually improves with each epoch. Fluctuations visible in epochs 10, 22, 30 and 44 in validation loss and GoF are not the result of instabilities, the zero-convergence problem as per Section 5 or overfitting, and are completely natural course of training. The test data predictions at these epochs do not deviate from the predictions from other epochs.

After 60-epoch-long training, the network is capable of removing artefacts and predicting actual structural information of objects, as shown in Figure 10a. In some cases, similarly to Figure 5, it treats some interfaces that are close to each other as one interface (see Figure 10b around pixels 120-140). The closer the interfaces are, the poorer the predictions get (Figure 10c).

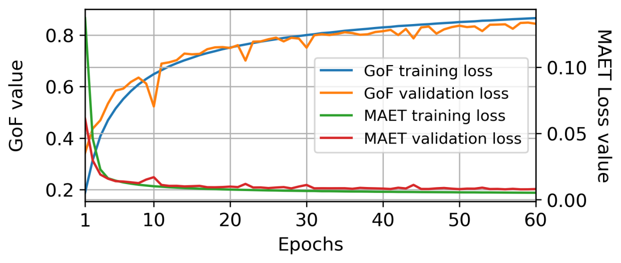

Figure 9. Model performance evaluation.

We show the performance of our model with the test dataset in Table 2 and Table 3. We use the default goodness-of-fit metrics parameters shown in Algorithms 1-3. We used optimised parameters from Section 9, and default parameters for the goodness of fit. Interestingly, there is not much difference between GoFAT and GoFUT, but, notably, predictions of values under the threshold are slightly better.

We observe better results for objects with a larger number of interfaces. As presented in Table 2, in such examples values of MAET are lower and GoF metrics are higher. When GoF is considered, the model has the biggest problem with predicting the structure of the simplest 2-interface objects as seen in Table 3. Such objects have the highest numbers

Table 2. Performance of the model for a test dataset. MAET (10^-3), GoF, GoFAT and GoFUT values are shown separately for each number of interfaces. The last column shows the mean value of the metrics. We present the mean MAET loss with its standard deviation in brackets. The best results are marked in bold.

Interfaces Metrics	2	3	4	5	6	7	8	9	10	11	12	All
MAET [10^-3]	9.7 (4.33)	8.66 (3.91)	8.56 (3.14)	8.14 (2.26)	7.98 (1.86)	7.81 (1.98)	7.71 (1.62)	7.4 (1.09)	7.36 (1.1)	7.37 (1.17)	**7.11 (1.1)**	7.99 (2.52)
GoF [%]	73.57 (6.05)	82.24 (5.03)	84.79 (4.29)	85.99 (3.44)	85.97 (3.34)	86.3 (3.54)	86.13 (2.98)	86.24 (2.73)	86.15 (2.8)	85.89 (2.76)	**86.39 (2.85)**	84.45 (5.32)
GoFAT [%]	76.92 (11.1)	80.5 (10.69)	80.29 (9.14)	81.82 (7.16)	82.25 (5.91)	83.31 (6.62)	83.38 (5.69)	84.92 (4.58)	85.55 (4.67)	85.99 (4.68)	**87.4 (4.6)**	82.88 (7.75)
GoFUT [%]	73.39 (6.24)	82.38 (5.0)	85.37 (4.24)	86.72 (3.35)	86.84 (3.38)	**87.1 (3.45)**	87.01 (2.85)	86.72 (2.98)	86.36 (3.0)	85.81 (3.01)	85.9 (3.09)	84.82 (5.52)

Table 3. The number of examples within the given GoF value range for a particular number of interfaces. Values inside the brackets represent the percentage of all examples for the given number of interfaces within the GoF range. The total values on the right show the number of all examples in the given GoF range and the percentage of all examples in the dataset. The total values at the bottom show how many examples there are with the given number of layers.

Interfaces GoF range [%]	2	3	4	5	6	7	8	9	10	11	12	Total
0 - 10	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
10 - 20	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
20 - 30	1 (0.21%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	1 (0.02%)
30 - 40	1 (0.21%)	1 (0.22%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	2 (0.04%)
40 - 50	0 (0%)	1 (0.22%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	1 (0.02%)
50 - 60	12 (2.46%)	1 (0.22%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	13 (0.26%)
60 - 70	89 (18.28%)	5 (1.12%)	7 (1.79%)	1 (0.2%)	2 (0.4%)	0 (0%)	0 (0%)	0 (0%)	1 (0.24%)	0 (0%)	1 (0.23%)	106 (2.12%)
70 - 80	357 (73.31%)	104 (23.21%)	25 (6.38%)	26 (5.26%)	19 (3.82%)	23 (5.4%)	15 (3.11%)	8 (1.55%)	9 (2.14%)	15 (3.7%)	9 (2.08%)	610 (12.2%)
80 - 90	27 (5.54%)	335 (74.78%)	343 (87.5%)	425 (86.03%)	441 (88.73%)	352 (82.63%)	423 (87.76%)	482 (93.41%)	394 (93.81%)	372 (91.85%)	393 (90.76%)	3987 (79.74%)
90 - 100	0 (0%)	1 (0.22%)	17 (4.34%)	42 (8.5%)	35 (7.04%)	51 (11.97%)	44 (9.13%)	26 (5.04%)	16 (3.81%)	18 (4.44%)	30 (6.93%)	280 (5.6%)
Total	487	448	392	494	497	426	482	516	420	405	433	5000

of data samples within GoF below 80% and never reach the range of 90-100%.

Results from Table 3 and Table 2 suggest that the examples with a larger number of interfaces are easier to interpret by the VGG16 models. It might be the consequence of more distinct and complex patterns in FFT stacks in such examples compared to simpler structures. It might seem that removing examples with a low number of interfaces from the training dataset will improve the prediction performance, but doing so will lead to failing in recognizing simpler structures since the network will not see their examples during training.

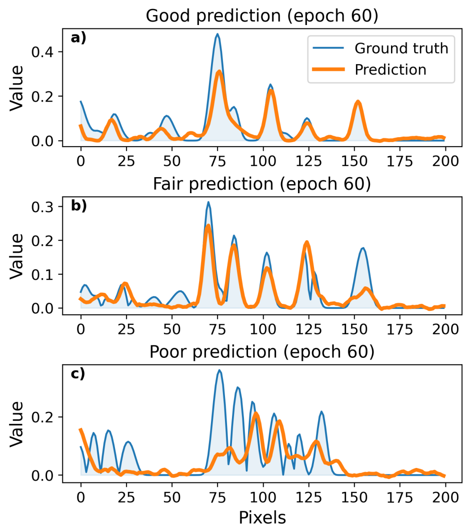

Figure 10. Ground truth (blue line) and predictions (orange lines) corresponding to each test data sample after the network was trained for 60 epochs. Plots were trimmed from 1024 pixels to 200 pixels to zoom into the relevant part of the test data.

10.2. Testing on a synthetic object

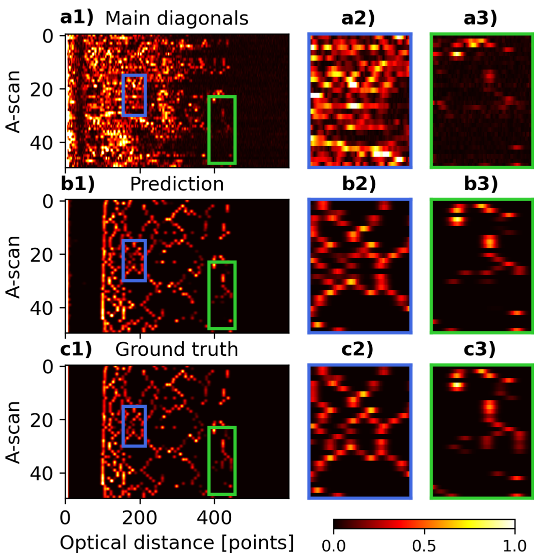

Figure 11. Computer-generated object representing an onion with noise. (a1) Image built from the middle row of FFT stacks (corresponding to the FFT of the main diagonals of the joint spectra). (b1) Image built from the predictions provided by our model. (c1) The ground truth image showing the actual structure of the onion. Green and blue show areas of interest where predictions do not match the ground truth are zoomed in in columns 2,3).

To demonstrate the performance of our model more qualitatively, we generated a small test dataset that contains 50 FFT stacks. It simulates the cellular structure of an onion. Its ideal noiseless image is presented in Figure 11 c1). In order to better match the data coming from an experimental setup, we incorporated noise at a level of 35dB into our input data. The central row of each noisy FFT stack (corresponding to the main diagonal in a noisy joint spectrum) is stacked one on top of another in Figure 11 a1) to generate an image. It can be seen that artefacts and noise cause a complete loss of the object's structure.

The image obtained by using the predictions of our model is presented in Figure 11 b1). Although the model was trained on noiseless data, and we introduced a high level of noise in the onion's joint spectra, our model performed very well in removing the artefacts and retrieving the structural information of the onion. We selected and zoomed in two areas of interest where our model struggled (Figure 11 columns 2,3). The loss of information in the blue area might be due to a high number of interfaces there. In the case of the green area (Figure 11a1), some information about the structure is lost due to noise. All in all, this example shows that our model provides, at least to some extent, immunity to noise present in the input signals. This may be because noise is random and does not generate patterns that the model is trained to recognise.

11. Conclusions

In this study, we proposed to use machine learning for removing artefacts from Q-OCT signals with 1024-by-1024-pixel-long joint spectra.

We showed that VGG16 is very good at recognising artefacts in the images. We presented that models trained on bigger datasets can perform better and provide better predictions than those trained on smaller ones, even when they return higher training MSE losses after training for the same amount of time. To mitigate the zero-convergence problem for sparse outputs, we introduced a new loss function. In addition, we created complementary goodness-of-fit metrics to measure the performance of regression models. We demonstrated that our approach allows for good performance even for data that incorporate noise, even though our model is trained on noiseless data.

Our method is very promising, but needs additional work. For example, it struggles with predicting the positions of interfaces in very simple objects as well as objects with multiple interfaces and small thicknesses. We consider several future directions. Firstly, we must modify and improve our loss function to be able to more closely detect interfaces. Also, architectures other than VGG should be tested to better differentiate the features in the inputs.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283, 2016.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.

Chollet, F. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.

De Boer, J. F., Leitgeb, R., and Wojtkowski, M. Twenty-five years of optical coherence tomography: the paradigm shift in sensitivity and speed provided by fourier domain oct. *Biomedical optics express*, 8(7):3248–3280, 2017.

Graciano, P. Y., Martínez, A. M. A., Lopez-Mago, D., Castro-Olvera, G., Rosete-Aguilar, M., Garduño-Mejía, J., Alarcón, R. R., Ramírez, H. C., and U'Ren, A. B. Interference effects in quantum-optical coherence tomography using spectrally engineered photon pairs. *Scientific Reports*, 9(1):1–14, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Iqbal, H. Harisiqbal88/plotneuralnet v1.0.0, December 2018. URL `https://doi.org/10.5281/zenodo.2526396`.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kolenderska, S. M. and Kolenderski, P. Intensity correlation oct – a true classical equivalent of quantum oct able to achieve up to 2-fold resolution improvement in standard oct images, 2021.

Kolenderska, S. M. and Szkulmowski, M. Artefact-removal algorithms for fourier domain quantum optical coherence tomography. *Scientific reports*, 11(1):1–8, 2021.

Kolenderska, S. M., Vanholsbeeck, F., and Kolenderski, P. Fourier domain quantum optical coherence tomography. *Optics Express*, 28(20):29576–29589, 2020a.

Kolenderska, S. M., Vanholsbeeck, F., and Kolenderski, P. Quantum optical coherence tomography using two photon joint spectrum detection (js-q-oct). *arXiv preprint arXiv:2005.13147*, 2020b.

Maliszewski, K. A. and Kolenderska, S. M. Artefact removal for quantum optical coherence tomography using machine learning. In *Optical Coherence Tomography and Coherence Domain Optical Methods in Biomedicine XXV*, volume 11630, pp. 1163012. International Society for Optics and Photonics, 2021.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.