`torchcvnn`: A PyTorch-based library to easily experiment with state-of-the-art Complex-Valued Neural Networks

Jérémy Fix[†], Quentin Gabot[*‡], X. Huy Nguyen[*‡],
Joana Frontera-Pons[‡], Chengfang Ren[*] and Jean-Philippe Ovarlez[*‡]
[*]SONDRA, CentraleSupélec, Université Paris-Saclay, 91190 Gif-sur-Yvette, France,
[†]LORIA, CNRS, CentraleSupélec, Université Paris-Saclay, F-57000 Metz, France,
[‡]DEMR, ONERA, Université Paris Saclay, F-91123 Palaiseau, France.

Abstract—**Complex-valued neural networks (CVNN) have attracted increasing attention in recent years, although their definition dates back to the mid-20th century. Indeed, several domains naturally process complex-valued signals, such as when sensing involves the response to an electromagnetic wave, such as remote sensing, MRI, etc. These domains would benefit from breakthroughs in complex-valued neural networks (CVNNs). We believe independent contributions to CVNNs must be gathered in a single, easy-to-use library. `torchcvnn` is an effort in that direction and provides several complex-valued building blocks, allowing us to experiment with CVNNs easily. The library is available at https://github.com/torchcvnn/torchcvnn alongside examples available at https://github.com/torchcvnn/examples.**

I. Introduction

Complex-valued Neural Networks (CVNNs) are a specialized type of artificial neural network where weights, inputs, activations, and outputs are represented by or process complex numbers. These networks are particularly well-suited for tasks involving complex-valued data or representations, e.g. Fourier transform and spectrogram. Although CVNNs have a long history [1], recent years have seen a resurgence of interest, particularly as their applications have begun to be extensively explored. Notable examples include their use in complex-valued data from synthetic aperture radar (SAR) [2]–[4] and MRI [5]. In the case of SAR systems, electromagnetic waves are transmitted, and the backscattered signals, characterized by both magnitude and phase, are measured. Additionally, CVNNs have gained attention for their potential in simulating bio-inspired spiked neural networks [6], particularly in applications related to neuronal synchrony for object discovery tasks [7].

Traditional real-valued neural networks can only process magnitude information, resulting in the loss of critical physical characteristics of the signals [8]. In contrast, CVNNs can fully utilize the complex nature of such data. This capability is essential in applications like Interferometric SAR (InSAR), where phase information is vital for extracting elevation data, and Polarimetric SAR (PolSAR), where understanding the physical and geometric properties of the observed area relies heavily on the phase information from polarization channels.

While the development of open-source frameworks like Theano, TensorFlow, and PyTorch has fueled rapid experimentation in real-valued neural networks, complex-valued neural networks have not received the same support in popular deep learning frameworks. Additional libraries are still needed to build upon these popular frameworks and include the necessary components for CVNNs. In TensorFlow and PyTorch, the optimization of complex-valued neural networks with real-valued loss is conducted using Wirtinger calculus. However, to effectively experiment with CVNNs, we also need various building blocks such as layers (e.g., fully connected, convolution, attention, batch normalization, etc.), activation functions, initializers, and access to standard dataset benchmarks. Recent work based on TensorFlow offers some of these elements [9]. At the same time, contributions in PyTorch are often scattered across various code repositories and sometimes rely on real-valued networks where the real and imaginary components are separated. Fortunately, now that PyTorch fully supports complex-valued differentiation, we can work on extensions that genuinely support complex-valued representations. Thus, we have started to develop `torchcvnn`, a library aiming to propose a PyTorch-like environment for CVNNs.

The objective of the `torchcvnn` library is to provide state-of-the-art models, including layers, activation functions, initialization functions, and standard datasets, to facilitate experimentation with complex-valued neural networks. The goal is to offer the community a modular library that simplifies the exploration of CVNNs.

The paper is organized as follows: in Section II, we briefly introduce the different components of the `torchcvnn` library, followed by different showcases on various tasks such as classification or segmentation of remote sensing data and neural implicit representations for MRI in Section III.

Notations: Italic type indicates a scalar quantity, lower case boldface indicates a vector quantity, and upper case boldface indicates a matrix or tensor. The conjugate operator is $\bar{.}$ and the transpose conjugate operator is ^{H}. $x \sim \mathcal{N}(\mu, \sigma^2)$ is a random Normal variable of mean μ and standard deviation σ, $x \sim \mathcal{U}(a, b)$ is a random uniform variable between a and b. The operator $\|.\|_2$ is the Frobenius norms operator, while $\|.\|_1$

is the l_1-norm. The operator $\odot$ is the Hadamard product. The operator ∇ is the Gradient differential operator. The symbol $\mathbf{1}$ is a matrix with all elements equal to 1. Finally, $\mathcal{F}(.)$ and $\mathcal{F}^{-1}(.)$ are the Fourier transform and the inverse Fourier transform, respectively.

II. THE COMPONENTS OF TORCHCVNN

In this section, we describe the components offered by `torchcvnn` to experiment with complex-valued neural networks. It essentially covers datasets and layers. Some of the layers require a specific implementation when working with complex-valued tensors. The others, such as upsampling, average pooling, and dropout, are provided by the library for technical reasons because the PyTorch framework (sometimes due to lower level implementations in CUDA) does not yet allow the application of these operations on complex-valued tensors (although, for example, averaging real numbers or complex numbers is the same operation). For these layers, `torchcvnn` reformulates the corresponding operation with the equivalent real-valued operation.

A. Datasets

`torchcvnn` offers classes for loading several datasets, some from the medical domain with cine MRI and some from the remote sensing domain. When the data are publicly available, a simple flag instructs `torchcvnn` to download the data. Illustrations of samples of the different datasets are given in Figure 1.

The API follows the usual PyTorch API for the datasets. The datasets usually expect the root directory of the dataset, possibly a flag to instruct downloading the data automatically, and a transform to be applied to the data once loaded. Additionally, some datasets support being split into tiles with a parametric overlap. This is a typical case for remote sensing datasets where the images are large and subdivided into smaller patches for further processing. As a matter of illustration, the code in Listing 1 loads a quad polarization SLC stack from the NASA Jet Lab UAV SAR mission and extracts the Pauli representation from the four polarizations.

```python
import numpy as np
import torchcvnn
from torchcvnn.datasets.slc.dataset import
    SLCDataset

def get_pauli(data):
    # Returns Pauli in (H, W, C)
    HH = data["HH"]
    HV = data["HV"]
    VH = data["VH"]
    VV = data["VV"]

    alpha = HH + VV
    beta = HH - VV
    gamma = HV + VH

    return np.stack([beta, gamma, alpha], axis=-1)


patch_size = (3000, 3000)
dataset = SLCDataset(
    rootdir,
    transform=get_pauli,
    patch_size=patch_size,
)
```

Listing 1: Example for loading a SLC stack and computing the Pauli representation of it.

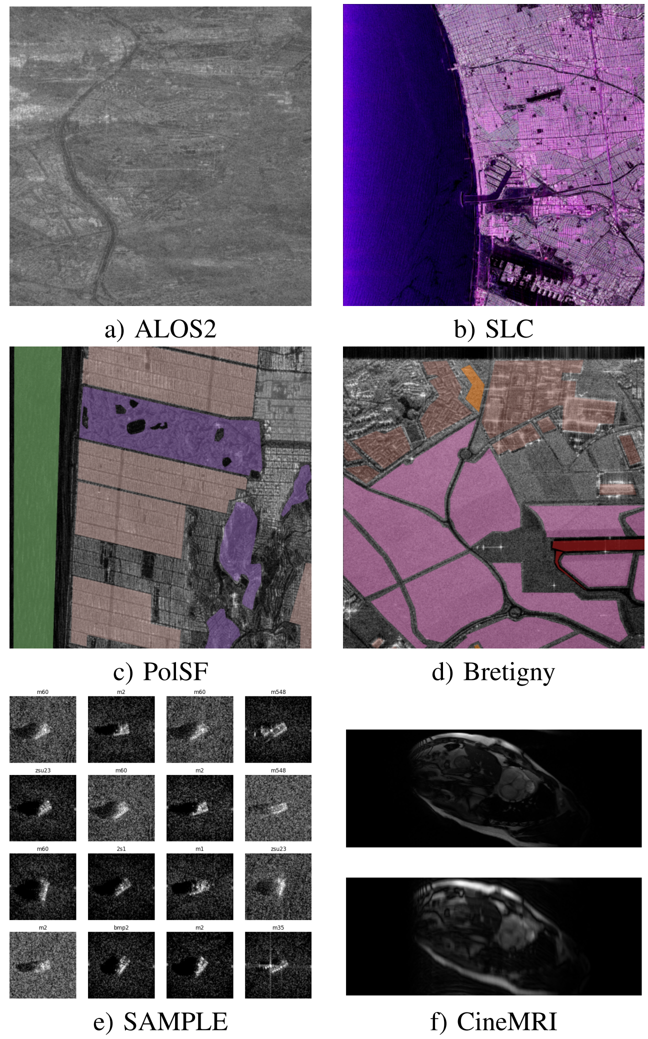


Fig. 1: Several datasets can be loaded with `torchcvnn` with data from remote sensing and MRI and for different tasks (classification or semantic segmentation). Unlabeled data in ALOS2 or SLC format can be loaded (a, b) as well as labeled data such as the PolSF (c) or Bretigny datasets. For the segmentation datasets, only one of the polarizations is displayed in magnitude (dB) with the labels overlaid. e) Classification dataset SAMPLE of 10 targets. f) An example of high (top) and low (bottom) resolution cardiac MRI images with a subsampling of the k-space using an acceleration factor of 8.

B. Activation functions

Several activation functions have been proposed in the literature. Split activation functions consist of a real-valued activation function applied independently to the real and imaginary components. They are implemented in `torchcvnn` with the generic class `IndependentRealImag` parameterized by the activation function to be applied. Fully complex activation functions jointly operate on complex numbers' real and imaginary components. Several activation functions are provided by `torchcvnn` such as CReLU, CPReLU, CSigmoid, zReLU, modReLU, Cardioid, Modulus, etc.

C. Initialization functions

Initialization of the neural networks is what triggered the revival of neural networks in the early 2000s with the design of pretraining algorithms (e.g., greedy layerwise pretraining) before the latter definition of appropriate initialization schemes such as the Glorot or He initializations. The performance of deep neural network training is critically dependent on them. The variance of the initialization schemes must be adapted for complex-valued neural networks, as shown in [10]. In `torchcvnn`, we follow the insights from [10]. Namely, for the Glorot Uniform/Normal and He Uniform/Normal, the bias is always initialized to 0.0, and the real and imaginary part of weights are generated randomly following the same distributions given by Equations (1)-(4):

$$\text{Glorot Uniform}: \quad \mathcal{U}\left[-\frac{\sqrt{3}}{\sqrt{\text{fan}_{\text{in}}+\text{fan}_{\text{out}}}}, \frac{\sqrt{3}}{\sqrt{\text{fan}_{\text{in}}+\text{fan}_{\text{out}}}}\right] \tag{1}$$

$$\text{Glorot Normal}: \quad \mathcal{N}\left(0, \frac{1}{\sqrt{\text{fan}_{\text{in}}+\text{fan}_{\text{out}}}}\right) \tag{2}$$

$$\text{He Uniform} \quad w \sim \mathcal{U}\left[-\frac{\sqrt{3}}{\sqrt{\text{fan}_{\text{in}}}}, \frac{\sqrt{3}}{\sqrt{\text{fan}_{\text{in}}}}\right] \tag{3}$$

$$\text{He Normal} \quad w \sim \mathcal{N}\left(0, \frac{1}{\sqrt{\text{fan}_{\text{in}}}}\right) \tag{4}$$

where fan_{in} and the fan_{out} denote respectively the number of inward connections and outward connections of a unit. Note that there is a factor $\frac{1}{\sqrt{2}}$ compared to real-valued initializers in terms of standard deviation. The main reason is for any random variable $Z = X + jY$ with independent and identically distributed real and imaginary parts, $\text{Var}(Z) = \text{Var}(X) + \text{Var}(Y) = 2\text{Var}(X)$, so the above factor maintains a unitary variance of the weights in the complex domain.

D. Normalization layers

Several normalization layers have been proposed in the literature. In [11], authors introduced batch normalization and demonstrated its efficiency in speeding up training by limiting the influence of early layers modifications on deep layers activations. The implementation in `torchcvnn` follows the adaptation to complex layers in [10] as given by Equation (5). It formulates complex batch normalization of scalars as the real-valued batch normalization of 2D vectors. This normalization and scaling is applied to every neuron of a layer by computing the empirical statistics, the mean vector $\boldsymbol{\mu}(\mathbf{x})$ and the covariance matrix $\boldsymbol{\Gamma}(\mathbf{x})$ from the total summed inputs of the neuron over a minibatch of samples:

$$\begin{aligned} \tilde{\mathbf{x}} &= \left(\boldsymbol{\Gamma}(\mathbf{x}) + \varepsilon \mathbf{I}\right)^{-\frac{1}{2}} \left(\mathbf{x} - \boldsymbol{\mu}(\mathbf{x})\right), \\ \hat{\mathbf{x}} &= \boldsymbol{\Lambda} \tilde{\mathbf{x}} + \boldsymbol{\beta}, \end{aligned} \tag{5}$$

where ε is a regularization term to ensure the invertibility of $\boldsymbol{\Gamma}(\mathbf{x})$ and where the trainable parameters are the complex offset $\boldsymbol{\beta}$ and 2×2 real-valued matrix $\boldsymbol{\Lambda}$.
Experimentally, we observed failure cases when using batch normalization with complex-valued neural networks. The activations may blow up when using the running average of the statistics, as is usual in evaluation mode. Our current guess is that this happens when the averaged covariance is close to degenerate.

In addition to batch normalization, other normalization layers such as Layer Norm [12] and RMS Norm [13] were introduced. Their implementation with complex tensors is very similar to batch normalization. The difference between batch normalization and layer normalization is that batch normalization computes its standardization statistics from the summed inputs to a unit over a mini-batch of samples. In contrast, the layer norm computes its statistics from the summed inputs of a layer's units for a single sample. RMSNorm is similar to LayerNorm, except it does not center its inputs, as given by Equation (6). It was shown in [13] that this centering can be removed, lowering the computational footprint without degrading the performances:

$$\begin{aligned} \tilde{\mathbf{x}} &= \left(\boldsymbol{\Gamma}(\mathbf{x}) + \varepsilon \mathbf{I}\right)^{-\frac{1}{2}} \mathbf{x}, \\ \hat{\mathbf{x}} &= \boldsymbol{\Lambda} \tilde{\mathbf{x}}. \end{aligned} \tag{6}$$

E. Pooling layers

Pooling layers were introduced in neural networks to spatially compress the representations and hopefully learn translation invariant representations. There are several possibilities to pool representations. Historical pooling layers are provided, such as *average* pooling and *m*ax pooling. For the *m*ax pooling, the *m*ax operation is computed from the module of the activations in the neuron's receptive fields. Pooling can also be performed following [14] with strided convolutions, as is now commonly applied.

F. Attention layers

Transformers were recently introduced in [15] for sequence processing, and extended to vision tasks with Vision Transformers (ViT) [16]. The extension to complex-valued tensors requires the adaptation of the multi-head attention layer. One head of the attention layer computes queries $\mathbf{Q}$, keys $\mathbf{K}$, and values $\mathbf{V}$ for input made of multiple tokens. With the real-valued neural network, for every token, the scaled dot product between its query $\mathbf{Q}$ and the keys $\mathbf{K}$ of all the tokens, normalized by a $\mathrm{softmax}$, determines weights to ponderate the values $\mathbf{V}$. In [17], it is suggested to replace the real-valued scaled dot product with the real part of the scaled complex-valued dot product as given by Equation (7) :

$$\mathbb{C}Att(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\text{Re}\left(\boldsymbol{Q}\,\boldsymbol{K}^{H}\right)}{\sqrt{d}}\right)\boldsymbol{V}\,. \qquad (7)$$

In this equation, the keys, queries and values computed for each token (or patch in a case of a ViT) are stacked in the rows of the matrices $\mathbf{K}, \mathbf{Q}, \mathbf{V}$, the softmax is applied on every rows independently and d corresponds to the embedding size of $\mathbf{Q}$ and $\mathbf{K}$. With the complex-valued dot product, the similarity between the queries and keys is proportional to their phase difference. With the complex-valued multi-head attention, one can implement complex-valued transformers and ViTs.

G. Models

There are two possibilities for building standard architectures with their complex-valued flavor. As shown in section III-A, one can benefit from real-valued neural networks and patch them to replace real layers with complex layers. Other models, such as ViT, are implemented directly within `torchcvnn`. Various scaled ViT are easily accessible with a few lines of code, as shown in Listing 2. In this example, the patch embedding module is a fully connected layer surrounded by RMSNorm layers. Using a convolutional layer in this example is a simple trick to split the input image into non-overlapping patches.

```

import torch.nn as nn
import torchcvnn.nn as c_nn
import torchcvnn.models as c_models

patch_embedder = nn.Sequential(
    c_nn.RMSNorm([cin, height, width]),
    nn.Conv2d(
        cin,
        hidden_dim,
        kernel_size=patch_size,
        stride=patch_size,
        dtype=torch.complex64,
    ),
    c_nn.RMSNorm([hidden_dim, height // patch_size,
    width // patch_size]),
)

vit_model = c_models.vit_b(patch_embedder)
# X is a torch tensor of dtype complex64
#                      and shape (B, C, H, W)
out = vit_model(X)
# out is of dtype complex64
# and shape
#    [B, hidden_dim, H//patch_size, W//patch_size]
```

Listing 2: Scaled ViT can be easily created with `torchcvnn` given a patch embedding neural network

Several scaled ViT are proposed with the hyperparameters given in table I. All of them use RMSNorm for the normalization layers.

Name	# Layers	# Heads	Hidden dim	FFN dim
vit_t	12	3	192	768
vit_s	12	6	384	1563
vit_b	12	12	768	3072
vit_l	24	16	1024	4096
vit_h	32	16	1280	5120

TABLE I: Parameters of predefined scaled ViTs, mirroring standard structures of real-valued VITs.

III. Showcasing torchcvnn with state of the art Complex-Valued Neural Networks

This section presents different use cases of `torchcvnn`. These examples highlight different facets of `torchcvnn`.

Class	A04	A05	A07	A10
Samples	573	573	573	567
Class	A32	A62	A63	A64
Samples	572	573	573	1417
Class	BTR_60	2S1	BRDM_2	D7
Samples	451	1164	1415	573
Class	SLICY	T62	ZIL131	ZSU_23_4
Samples	2539	572	573	1401

TABLE II: MSTAR dataset class names and samples per class.

In subsection III-A, we show how to implement complex-valued neural networks either by "patching" real-valued neural networks, replacing real-valued building blocks with complex-valued counterparts, or using state-of-the-art models such as Vision Transformers [16], using the complex-valued multi-head attention of [17]. In subsection III-B, we consider an encoder/decoder architecture for reconstructing remote sensing data. This encoder/decoder example is further refined in section III-C, where a complex-valued UNet is used for a semantic segmentation task of SAR images. Finally, in subsection III-D, we show how `torchcvnn` can help implement an Implicit Neural Representation in the context of cardiac MRI reconstruction.

A. Image classification with Complex-valued Neural Networks and Vision Transformers

Experimental datasets: In the field of Synthetic Aperture Radar (SAR), the MSTAR (Moving and Stationary Target Acquisition and Recognition) [18] dataset is a standard resource for SAR image analysis. MSTAR is widely used in remote sensing imaging research, and is developed and publicly released by the US Defense Advanced Research Projects Agency (DARPA). The dataset includes complex-valued SAR images of various military vehicles, such as tanks, armored carriers, trucks, or utility conveyances. Images are taken by the Sandia X-band SAR platforms in spotlight mode, providing a one-foot range resolution. Targets are captured from multiple azimuth angles, from 0° to 360°. The number of classes and distribution of samples per class are given in table II. Note this dataset is slightly unbalanced.

Data preprocessing: Note that the image size of MSTAR dataset vary from 54×54 to 193×193. All the images have been resized by either zero-padding or center-cropping their Discrete Fourier Transform. The magnitudes have been converted to dB keeping the phase unchanged. The data are split in 80% for training and 20% for testing.

Models	# Layers	# Heads	Hidden dim
ViT	3	8	128
Conv-ViT	3	8	256

TABLE III: Two ViT architectures are tested. For the Conv-ViT, the input tensor goes through a Convolutional stem which is made of two Conv-BN-modRelu blocks.

Models	Params	Inp. size	Top-1 Acc.	Top-5 Acc
ResNet-18	11.2M	128	99.8%	100%
ViT	110K	128	81.2%	99.3%
ViT	116K	208	89.8%	99.8%
Conv-ViT	695K	128	87.5%	99.7%
Conv-ViT	709K	208	91.1%	99.8%

TABLE IV: Experimental training results. Models details are in Table III.

Models: To construct state-of-the-art complex-valued vision neural networks, one can use already implemented real-valued neural networks and "patch them" in order to replace real-valued modules by their complex-valued counterparts. A wide range of models are available either built in torchvision or with external librairies such as Timm [19]. The Listing 3 illustrates how to convert a real-valued network into a complex-valued network by patching it.

Another approach is obviously to implement your own neural network. As explained in section II-G, `torchcvnn` provides several scaled ViTs which are getting popular in recent years to address computer vision problems. While vanilla Vision Transformers deliver remarkable performances on large-scale datasets, the lack of inductive bias prone them to easily and rapidly overfit on smaller datasets. We have designed a complex-valued lightweight ViT for better fitting on the MSTAR dataset with 3 layers, 8 attentional heads with hidden dimension of either 128 or 256 (see Table III). In these experiments, we considered two variations denoted as ViT and Conv-ViT where the Conv-ViT involves a convolutional block before applying the ViT.

Training details: Training is performed with AdamW with a learning rate of 0.003, a weight decay of 0.05 for the ResNet and 0.03 for the ViT. The learning rate is halved if the validation loss is not improved over 8 epochs. Batch size is 64 for the ResNet and 128 for the ViT. For both baseline ViTs and hybrid-ViTs models, input images are divided into 16x16 patches.

Results: Our best ViT involves the convolutional stem (see Table IV) and achieves 91% of top-1 accuracy. Overall, the patched ResNet-18 performs better compared to the ViTs but further experiments may identify better hyperparameters allowing the ViTs to be more competitive. It is also possible that MSTAR is too small to prevent the ViTs from overfitting. We also tested several input image sizes. The performances of ResNet-18 did not improve significantly with sizes larger than 128×128. However, the ViT performances improved by resizing the images to 208×208. The code is available at https://github.com/torchcvnn/examples/tree/main/mstar_classification.

```

import torch.nn as nn
import torchcvnn.nn as c_nn
from torchvision.models import resnet18

def convert_to_complex(module: nn.Module) -> nn.Module:
    cdtype = torch.complex64
    for name, child in module.named_children():
        if isinstance(child, nn.Conv2d):
            setattr(
                module,
                name,
                nn.Conv2d(
                    child.in_channels,
                    child.out_channels,
                    child.kernel_size,
                    stride=child.stride,
                    padding=child.padding,
                    bias=child.bias is not None,
                    dtype=cdtype,
                ),
            )
        elif isinstance(child, nn.ReLU):
            setattr(module, name, c_nn.modReLU())
        elif isinstance(child, nn.BatchNorm2d):
            setattr(module, name, c_nn.BatchNorm2d(child.num_features))
        ....
        else:
            convert_to_complex(child)

complex_valued_model = convert_to_complex(resnet18())
```

Listing 3: Example function for patching a real-valued into a complex-valued neural network with `torchcvnn`

B. Remote sensing reconstruction with complex-valued auto-encoder

The example of this section is a replicate of [8]. Complex-valued neural networks are widely used to process complex-valued data, especially when the phase information carries valuable information. In the case of polarimetric SAR imaging, the polarization properties of electromagnetic waves are used to measure the polarimetric properties of scatterers, using both amplitude and phase information. A complex scattering matrix is acquired for each pixels of the image:

$$\mathbf{S} = \begin{pmatrix} S_{HH} & S_{VH} \\ S_{HV} & S_{VV} \end{pmatrix}, \tag{8}$$

by considering the horizontal and vertical polarization basis. Thus, a PolSAR image is a complex-valued tensor $\in \mathbb{C}^{4 \times h \times w}$ where h, w are the image height and width.
From this scattering matrix, various decompositions have been proposed. Coherent decompositions, like Pauli [20] and Krogager [21]–[23], rely on the expression of $\mathbf{S}$ as a combination of the backscattering mechanisms response of canonical objects. On the other hand, non-coherent decompositions, like $H - \alpha$ [24], a covariance matrix is estimated locally for each pixel to analyze random scattering effects better.
Gabot et al. [8] demonstrated that polarimetric properties were

mainly preserved after a complex-valued AutoEncoder reconstruction process. They notably showed good performances in the confusion matrix between original and reconstructed $H-\alpha$ classes, as seen in Figures 2 and 3. A complex-valued auto-encoder requires both the down and up sampling blocks. As shown ni Listing 4, implementing these blocks with `torchcvnn` is straighforward by combining the complex-valued convolution layers already implemented by PyTorch and adding the `torchcvnn` normalization and activation layers.

```python

import torch
import torchcvnn.nn as c_nn

def up(cin, cmid, cout, dtype=torch.complex64):
    conv_params = {"kernel_size": 3, "padding": 1}
    return nn.Sequential([
        nn.Conv2d(cin, cmid, stride=2,
                  **conv_params, dtype=dtype),
        c_nn.BatchNorm(cmid),
        c_nn.modReLU(),
        nn.Conv2d(cmid, cout, stride=1,
                  **conv_params, dtype=dtype),
        c_nn.BatchNorm(cout),
        c_nn.modReLU(),
    ])

def down(cin, cout):
    conv_params = {"kernel_size": 3,
                   "padding": 1,
                   "stride": 1}
    return nn.Sequential([
        c_nn.ConvTranspose2d(
            cin, cout, kernel_size=2, stride=2
        ),
        nn.Conv2d(cout, cout,
                  **conv_params, dtype=dtype),
        c_nn.BatchNorm(cout),
        c_nn.modReLU(),
        nn.Conv2d(cout, cout,
                  **conv_params, dtype=dtype),
        c_nn.BatchNorm(cout),
        c_nn.modReLU(),
    ])
```

Listing 4: Example autoencoder down- and up- scaling modules with `torchcvnn`

C. Semantic segmentation with complex-valued U-Net

Semantic segmentation is a significant task in computer vision, even when considering complex-valued data like SAR imaging. The assignment of each pixel to a certain class is usually achieved with the use of the U-Net architecture [25] or one of its many variants [26]–[28]. Most of these models have been extended to the complex domain for semantic segmentation of SAR images. Barrachina et al. proposed an implementation of a complex-valued U-Net in [29] on the PolSF dataset. In contrast, Wang et al. [30] implemented a complex-valued U-Net to suppress nonhomogeneous sea clutter.

In this section, we showcase the performances of a custom complex-valued UNet using `torchcvnn`. The different parts of the model (convolutional layers, convolutional transpose layers, activation functions) have been designed using the library, while the training and evaluation have been made on the integrated PolSF dataset. Examples codes for the upsampling and downsampling modules are given in Listing 4. The model uses the modReLu activation function. The encoder is built from 5 modules, each being composed of residual blocks with two stacked Conv-BatchNorm-modReLU using a kernel size of 3. The decoder is built from a sequence of 5 modules, each being composed of an upsampling, a concatenation of the corresponding encoder features, followed by the same residual blocks sequence as the encoder. The model has a total of 52 million trainable parameters. The model has been trained on a segmentation task, using a custom Focal loss implementation with AdamW with a learning rate $\epsilon = 0.001$ and a weight decay of 0.005. The learning rate is adapted with a cosine annealing with warm restart strategy. We obtained performances that were close to state-of-the-art, as displayed in Figure 4 & 5.

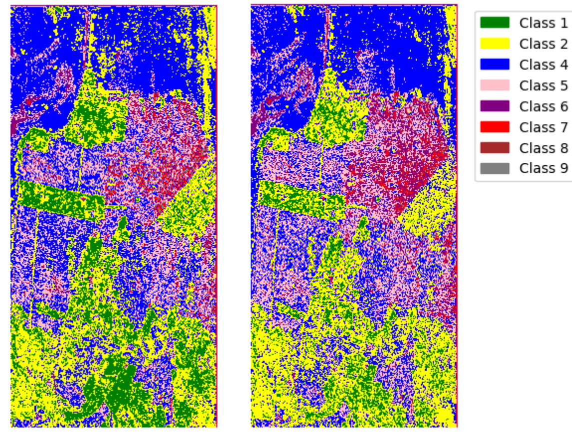
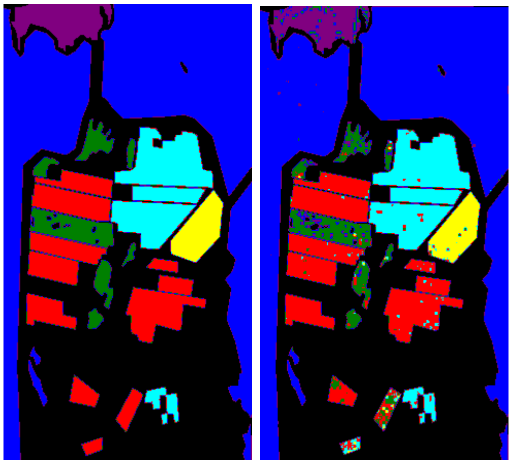


Fig. 2: Images of the original (left) and reconstructed (right) images with the $H-\alpha$ classification.

Fig. 3: Confusion matrix of the original (rows) and reconstructed (columns) $H-\alpha$ classes.

D. Complex-Valued Implicit Neural Representations

In this section, we present the use of complex-valued neural networks for implicit neural representations applied to cardiac reconstruction following [5]. The code for this example is



(a) (b)

Fig. 4: Images of the ground truth (a) and predicted (b) segmentation map. Each of the 6 classes is represented with its own color, black pixels belonging to the ”unlabeled” class. A mask is applied on the predicted segmentation map to replicate the ”unlabeled” class.

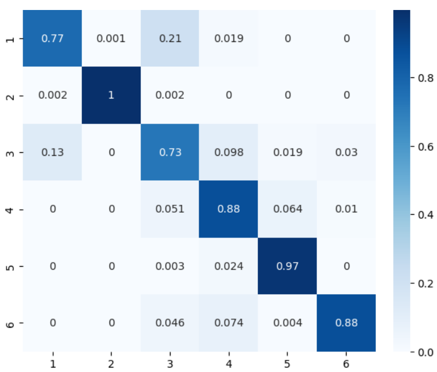
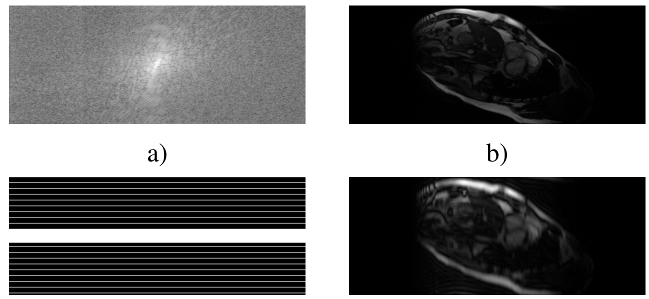


Fig. 5: Confusion matrix of the ground truth (rows) and predicted (columns) classes over the labeled classes.

available at https://github.com/torchcvnn/examples/tree/main/nir_cinejense.

The Implicit Neural Representation (INR) approach has recently emerged as a powerful technique for approximating functions in various fields [31]. It is based on the optimization of a function $f_\theta(z)$ that can be queried at some coordinates z and is parametrized by θ. The coordinates can be arbitrary, for example, two-dimensional when interpolating images [32], and three-dimensional when interpolating a volume [33].

Let us briefly introduce the context of cardiac image reconstruction before explaining how `torchcvnn` helps to reproduce the work [5]. Cardiac magnetic resonance imaging (CMR) is a non-invasive imaging technique that allows the heart to be observed through time. One difficulty is the slow speed of the process, which leads to patient discomfort and motion artifacts. If imaging is carried out for an extended period, the resolution of the heart is better. Hence, the challenge is to shorten the observation while still being able to interpolate the heart with a high resolution. This question is the aim of the CMRxRecon challenge https://cmrxrecon.github.io hosted in the context of the International Conference on Medical Image Computing and Computer Assisted Intervention, MICCAI 2023. In this challenge, the heart's Fourier representation (so-called k-space) is partially observed, and the task is to reconstruct it fully. Bands subsample the Fourier representation, and depending on the acceleration factor (4, 8, or 10), more or fewer frequency bands of the signals are provided. The illustrations of the subsampled k-space and the resulting heart images are given in Figure 6 alongside the fully sampled k-space and full-resolution image.



a) b) c) d)

Fig. 6: a) Fully sampled k-space of a specific slice, coil and frame. The k-space is plotted in dB. b) Reconstructed heart image using the root sum of squares of the inverse Fourier transform of each coil. c) Subsampling mask for an acceleration factor of 10. d) Reconstructed heart image using the root sum of squares of the inverse Fourier transform of the subsampled k-spaces of each coil.

Several approaches can be considered to address this challenge, as presented during the MICCAI 2023 conference [34]. We selected the approach of [5] to showcase how to use `torchcvnn` to train complex-valued INR. The approach of CineJENSE [5] is a 2D+t extension of the earlier work IMJENSE of [35], which introduced INR for cardiac image reconstruction. Borrowing the notations from [5], MR imaging reconstruction aims to find the image $\mathbf{I} \in \mathbb{C}^{N_x \times N_y}$ from the partially observed k-space $\mathbf{K} \in \mathbb{C}^{N_x \times N_y \times N_c}$, where $N_x \times N_y$ denotes the image dimensions and N_c the number of coils used for the imaging. The link between the image space and the k-space is through the operator $\mathbf{F}(.)$, involving the sensitivities of the coils, the Fourier transform $\mathcal{F}$, and a subsampling mask of the k-space. The reconstruction is there formulated as the minimization problem (9):

$$\min_{\mathbf{I},\mathbf{S}} \|\mathbf{F}(\mathbf{I}) - \mathbf{K}\|_2^2 + \lambda_I \, R(\mathbf{I}) + \lambda_S \, R(\mathbf{S}) \,, \tag{9}$$

where $R(\mathbf{I})$ and $R(\mathbf{S})$ are two regularization terms, respectively, on the image $\mathbf{I}$ and the sensitivity maps $\mathbf{S}$ and their regulation parameters λ_I and λ_S. The input coordinates of the INR are 2D+t spatiotemporal (x, y, t). Although earlier works on INR directly used the raw coordinates as inputs to the neural networks, later research proposed other schemes with improved performances. Notably, [32] introduced a multi-resolution hash encoding scheme, which is involved before feeding the coordinates into a multi-layer feedforward neural network. In the current implementation, the tiny-cuda-nn [36] implementation is used. It outputs real-valued embeddings that are then fed into a complex-valued neural network. In a future release, `torchcvnn` will support complex-valued encodings.

Two INR networks are trained, one for the image at instant $t \in [0, T-1]$, denoted $\mathbf{I}^t_\theta(x, y)$ and one for the coil's sensitivity, denoted $\mathbf{S}^{t,c}_\psi(x, y)$. The output of these two networks is multiplied to obtain the multi-coil k-space $\mathbf{K}^{t,c}_{\theta,\psi}$. As stated earlier, the multi-coil k-space is not fully observed, and only a fraction of it is provided; this fraction is defined by the binary mask $\mathbf{M}$. To estimate the parameters θ and ψ, the criterion to be minimized combining all these elements, is then defined as:

$$\left(\hat{\theta}, \hat{\psi}\right) = \underset{\theta,\psi}{\operatorname{argmin}} \frac{1}{N_c T} \sum_{\substack{c=0\\t=0}}^{\substack{T-1\\N_c-1}} L_\delta\left(\mathbf{M} \odot \mathcal{F}\left(\mathbf{I}^t_\theta \odot \mathbf{S}^{t,c}_\psi\right), \mathbf{K}^{t,c}\right) + \lambda \|\nabla \mathbf{I}^{t,c}_\theta\|_1 \,, \tag{10}$$

where λ is a regularization parameter and L_δ is the Huber loss [37] with the threshold $\delta = 1$. The first term of (10) is the minimization between the predicted and observed k-space, while the second term is the total variation loss in the image space acting as a regularizer. Minimizing the loss has the joint consequence of optimizing the image and the sensitivity maps. During inference, the predictions of the two neural networks are used to fill in only the unobserved components of the k-space. The final reconstructed image $\hat{\mathbf{I}}^t$, at time t, combining all the N_c coils, hence reads:

$$\hat{\mathbf{I}}^t = \sum_{c=0}^{N_c-1} \overline{\mathbf{S}}^{t,c}_\psi \odot \mathcal{F}^{-1}\Big((\mathbf{1} - \mathbf{M}) \odot \mathcal{F}\Big(\mathbf{I}^t_{\hat{\theta}} \odot \mathbf{S}^{t,c}_{\hat{\psi}}\Big) + \mathbf{M} \odot \mathbf{K}^{t,c}\Big) \,. \tag{11}$$

The reconstruction quality is evaluated with the Peak Signal-to-Noise Ratio (PSNR) and Structural SIMilarity (SSIM). The models were trained for 256 iterations for the experiments reported in this article, optimized with Adam and learning rate $\epsilon = 0.01$. The total variation regularization factor is set to $\lambda = 4.0$. The parameters for the two hash grids are the same as in [5]. The two feedforward neural networks have 4 hidden layers with modReLU activation.

The performances obtained are provided in Tables V-VII for the three acceleration factors and two views (SAX/LAX). They are comparable to [5], although slightly better than those reported in the paper by the authors. These metrics involve all the 109 data for the SAX view and 94 for the LAX view. There are no separate train/valid folds because the INR is trained individually for every sample. Hence, the metrics reported in Tables V-VII are real risk estimates. The zero filled baseline is computed by setting to 0 the unobserved parts of the k-space.

View	Method	SSIM	PSNR (db)
SAX	Zero Filled	0.83 ± 0.04	29.42 ± 1.49
	CineJENSE	0.96 ± 0.01	$42.19 \pm 2..13$
LAX	Zero Filled	0.81 ± 0.03	28.47 ± 1.59
	CineJENSE	0.97 ± 0.01	42.93 ± 1.72

TABLE V: Performance for acceleration factor $R = 4$.

View	Method	SSIM	PSNR (db)
SAX	Zero Filled	0.81 ± 0.04	28.50 ± 1.62
	CineJENSE	0.89 ± 0.02	35.61 ± 1.67
LAX	Zero Filled	0.80 ± 0.04	27.73 ± 1.59
	CineJENSE	0.90 ± 0.02	35.43 ± 1.82

TABLE VI: Performance for acceleration factor $R = 8$.

View	Method	SSIM	PSNR (db)
SAX	Zero Filled	0.81 ± 0.04	28.24 ± 1.59
	CineJENSE	0.88 ± 0.02	34.35 ± 1.62
LAX	Zero Filled	0.79 ± 0.03	27.52 ± 1.62
	CineJENSE	0.89 ± 0.02	34.42 ± 1.62

TABLE VII: Performance for acceleration factor $R = 10$.

IV. Perspectives

The keen interest in deep learning has fueled the scientific community with an ever-growing number of new tasks and methods. Open source initiatives like PyTorch have played a crucial role in democratizing deep learning by proposing frameworks, datasets, and pre-trained models. This approach relies on the active participation of the scientific community, from the conception of new methods to their actual implementation. In this article, we showcased `torchcvnn`, an open-source framework for complex-valued deep learning inspired by the PyTorch initiative. `torchcvnn` proposes various modules and datasets for the community to use in their experiments. We have notably highlighted how some crucial parts of complex-valued neural networks are implemented (activation functions, attention layers, normalization layers) while presenting the available complex-valued datasets (ALOS2, SAMPLE, MRI). To demonstrate the capacities of `torchcvnn`, we have illustrated some applications on various key tasks (classification, segmentation, reconstruction), using the library to build, train, and evaluate the model. Indeed, `torchcvnn` eases the design of new complex-valued neural networks, as well as extend SOTA models (CNN, ViT, NIR) to the complex case. We plan to add several datasets (S1SLC, fastMRI), as well as complex-valued embeddings (ROPE, hash encoding) and transfer functions (elementary transcendental functions). In addition, several applications could benefit from `torchcvnn` as the backbone of their implementations. Finally, increasing contributions are expected in the hope of establishing an active open-source community.

References

[1] A. Hirose, *"Complex-Valued Neural Networks"*. Springer, 2012.

[2] Y. Cao, Y. Wu, P. Zhang, W. Liang, and M. Li, "Pixel-wise PolSAR image classification via a novel complex-valued deep fully convolutional network," *Remote Sensing*, vol. 11, no. 22, p. 2653, 2019.

[3] J. Zhao, M. Datcu, Z. Zhang, H. Xiong, and W. Yu, "Contrastive-regulated CNN in the complex domain: A method to learn physical scattering signatures from flexible PolSAR images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 12, pp. 10 116–10 135, 2019.

[4] V. Dhédin, J. Levi, J. Fix, C. Ren, and I. Hinostroza, "Complex-valued wasserstein gan for sar image generation," in *IGARSS 2024 - 2024 IEEE International Geoscience and Remote Sensing Symposium*, 2024, pp. 6991–6996.

[5] Z. Al-Haj Hemidi, N. Vogt, L. Quillien, C. Weihsbach, M. P. Heinrich, and J. Oster, "Cinejense: Simultaneous cine MRI image reconstruction and sensitivity map estimation using neural representations," in *Statistical Atlases and Computational Models of the Heart. Regular and CMRxRecon Challenge Papers*, O. Camara, E. Puyol-Antón, M. Sermesant, A. Suinesiaputra, Q. Tao, C. Wang, and A. Young, Eds. Cham: Springer Nature Switzerland, 2024, pp. 467–478.

[6] D. P. Reichert and T. Serre, "Neuronal synchrony in complex-valued deep networks," 2014. [Online]. Available: https://arxiv.org/abs/1312.6115

[7] S. Löwe, P. Lippe, M. Rudolph, and M. Welling, "Complex-valued autoencoders for object discovery," *Transactions on Machine Learning Research*, 2022. [Online]. Available: https://openreview.net/forum?id=1PfcmFTXoa

[8] Q. Gabot, J. Fix, J. Frontera-Pons, C. Ren, and J.-P. Ovarlez, "Preserving polarimetric properties in PolSAR image reconstruction through Complex-Valued Auto-Encoders," in *RADAR 2024*, Rennes, France, Oct. 2024. [Online]. Available: https://hal.science/hal-04785702

[9] J. A. Barrachina, C. Ren, G. Vieillard, C. Morisseau, and J.-P. Ovarlez, "Theory and implementation of complex-valued neural networks," 2023. [Online]. Available: https://arxiv.org/abs/2302.08286

[10] C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal, "Deep complex networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=H1T2hmZAb

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 448–456. [Online]. Available: https://proceedings.mlr.press/v37/ioffe15.html

[12] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016. [Online]. Available: https://arxiv.org/abs/1607.06450

[13] B. Zhang and R. Sennrich, "Root Mean Square Layer Normalization," in *Advances in Neural Information Processing Systems 32*, Vancouver, Canada, 2019. [Online]. Available: https://openreview.net/references/pdf?id=S1qBAf6rr

[14] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," in *International Conference on Learning Representations*, 2015.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6000–6010.

[16] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[17] F. Eilers and X. Jiang, "Building blocks for a complex-valued transformer architecture," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023, pp. 1–5.

[18] J. R. Diemunsch and J. Wissinger, "Moving and stationary target acquisition and recognition (MSTAR) model-based automatic target recognition: Search technology for a robust ATR," in *Algorithms for synthetic aperture radar Imagery V*, vol. 3370. SPIE, 1998, pp. 481–492.

[19] R. Wightman, N. Raw, A. Soare, A. Arora, C. Ha, C. Reich, F. Guan, J. Kaczmarzyk, mrT23, Mike, SeeFun, contrastive, M. Rizin, H. Kim, C. Kertész, D. Mehta, G. Cucurull, K. Singh, hankyul, Y. Tatsunami, A. Lavin, J. Zhuang, M. Hollemans, M. Rashad, S. Sameni, V. Shults, Lucain, X. Wang, Y. Kwon, and Y. Uchida, "rwightman/pytorch-image-models: v0.8.10dev0 release," Feb. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7618837

[20] V. Alberga, E. Krogager, M. Chandra, and G. Wanielik, "Potential of coherent decompositions in SAR polarimetry and interferometry," in *IGARSS 2004. 2004 IEEE International Geoscience and Remote Sensing Symposium*, vol. 3. IEEE, 2004, pp. 1792–1795.

[21] E. Krogager, "Utilization and interpretation of polarimetric data in high resolution radar target imaging," in *Proc. Second International Workshop on Radar Polarimetry (JIPR'1992)*, vol. 2, Nantes, France, Sep. 8–10, 1992, pp. 547–557.

[22] E. Krogager and Z. H. Czyz, "Properties of the sphere, diplane, helix decomposition," in *Proc. Third International Workshop on Radar Polarimetry (JIPR'1995)*, vol. 1, Nantes, France, Mar. 21–23, 1995, pp. 106–114.

[23] E. Krogager, J. Dall, and S. N. Madsen, "The sphere, diplane, helix decomposition recent results with polarimetric SAR data," in *Proc. Third International Workshop on Radar Polarimetry (JIPR'1995)*, vol. 2, Nantes, France, Mar. 21–23, 1995, pp. 621–625.

[24] P. Formont, F. Pascal, G. Vasile, J.-P. Ovarlez, and L. Ferro-Famil, "Statistical classification for heterogeneous polarimetric SAR images," *IEEE Journal of selected topics in Signal Processing*, vol. 5, no. 3, pp. 567–576, 2010.

[25] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer, 2015, pp. 234–241.

[26] O. Oktay, J. Schlemper, L. L. Folgoc, M. Lee, M. Heinrich, K. Misawa, K. Mori, S. McDonagh, N. Y. Hammerla, B. Kainz *et al.*, "Attention u-net: Learning where to look for the pancreas," *arXiv preprint arXiv:1804.03999*, 2018.

[27] M. Z. Alom, M. Hasan, C. Yakopcic, T. M. Taha, and V. K. Asari, "Recurrent residual convolutional neural network based on u-net (r2u-net) for medical image segmentation," *arXiv preprint arXiv:1802.06955*, 2018.

[28] S. Jégou, M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio, "The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017, pp. 11–19.

[29] J. A. Barrachina, C. Ren, G. Vieillard, C. Morisseau, and J.-P. Ovarlez, "Real-and complex-valued neural networks for SAR image segmentation through different polarimetric representations," in *2022 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2022, pp. 1456–1460.

[30] Y. Wang, W. Zhao, X. Wang, J. Chen, H. Li, and G. Cui, "Nonhomogeneous sea clutter suppression using complex-valued u-net model," *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1–5, 2022.

[31] Y. Xie, T. Takikawa, S. Saito, O. Litany, S. Yan, N. Khan, F. Tombari, J. Tompkin, V. Sitzmann, and S. Sridhar, "Neural fields in visual computing and beyond," *Computer Graphics Forum*, 2022.

[32] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, vol. 41, no. 4, pp. 102:1–102:15, Jul. 2022. [Online]. Available: https://doi.org/10.1145/3528223.3530127

[33] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: representing scenes as neural radiance fields for view synthesis," *Commun. ACM*, vol. 65, no. 1, p. 99–106, Dec. 2021. [Online]. Available: https://doi.org/10.1145/3503250

[34] O. Camara, E. Puyol-Antón, M. Sermesant, A. Suinesiaputra, Q. Tao, C. Wang, and A. A. Young, Eds., *Statistical Atlases and Computational Models of the Heart. Regular and CMRxRecon Challenge Papers - 14th International Workshop, STACOM 2023, Held in Conjunction with MICCAI 2023, Vancouver, BC, Canada, October 12, 2023, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 14507. Springer, 2024. [Online]. Available: https://doi.org/10.1007/978-3-031-52448-6

[35] R. Feng, Q. Wu, J. Feng, H. She, C. Liu, Y. Zhang, and H. Wei, "Imjense: Scan-specific implicit representation for joint coil sensitivity and image estimation in parallel MRI," *IEEE Transactions on Medical Imaging*, vol. 43, no. 4, pp. 1539–1553, 2023.

[36] T. Müller, "tiny-cuda-nn," April 2021. [Online]. Available: https://github.com/NVlabs/tiny-cuda-nn

[37] P. J. Huber, "Robust Estimation of a Location Parameter," *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73 – 101, 1964. [Online]. Available: https://doi.org/10.1214/aoms/1177703732