Deep Quaternion Features for Privacy Protection

Hao Zhang [*1] **Yiting Chen** [*1] **Haotian Ma** [2] **Liyao Xiang** [1] **Jie Shi** [3] **Quanshi Zhang** [1]

Abstract

We propose a method to revise the neural network to construct the quaternion-valued neural network (QNN), in order to prevent intermediate-layer features from leaking input information. The QNN uses quaternion-valued features, where each element is a quaternion. The QNN hides input information into a random phase of quaternion-valued features. Even if attackers have obtained network parameters and intermediate-layer features, they cannot extract input information without knowing the target phase. In this way, the QNN can effectively protect the input privacy. Besides, the output accuracy of QNNs only degrades mildly compared to traditional neural networks, and the computational cost is much less than other privacy-preserving methods.

1. Introduction

It is often the case that the raw data collected for deep learning cannot be processed locally, so that features are extracted and processed elsewhere. The privacy leakage of those features have been receiving wide attention recently. Many works such as (Dosovitskiy & Brox, 2016; Zeiler & Fergus, 2014; Mahendran & Vedaldi, 2015; Shokri et al., 2017; Ganju et al., 2018; Melis et al., 2018; Zhang et al., 2019) have pointed out that, attackers can recover significant amount of sensitive input information from the intermediate-layer features of a DNN. As countermeasures, several studies (Osia et al., 2017; Li et al., 2017; Wang et al., 2018; Zhang et al., 2016; Mohassel & Zhang, 2017) have been proposed, but have limited applicability mostly due to the exorbitant computational cost.

The task of attribute obfuscation and privacy protection on deep neural networks mainly propose the following two requirements: First, even if attackers have obtained network parameters and intermediate-layer features, they should not be able to reconstruct the input or infer private attributes of inputs. Second, the privacy-protection method is not supposed to increase the computational cost, or affect the task accuracy significantly.

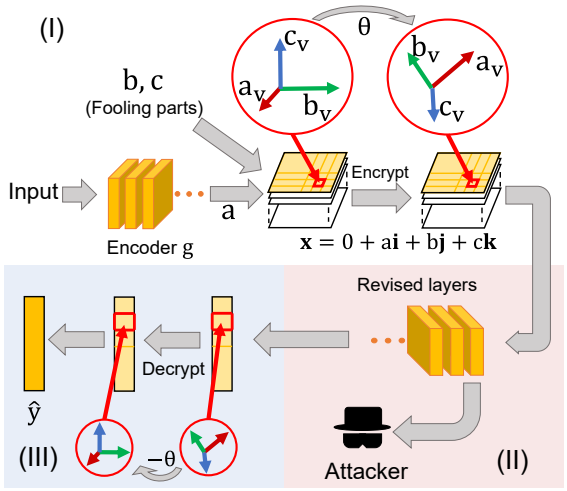

Figure 1. Overview of the QNN. The encoder module (I) is located at the local device. The traditional real-valued feature a is extracted and converted into a quaternion-valued tensor for encryption. The quaternion-valued feature is rotated by θ. The rotated feature is processed by the processing module (II) which satisfies the rotation equivariance property. The decoder module (III) uses θ to decrypt features to get the final result.

Therefore, we propose a generic method to revise a traditional neural network to a quaternion-valued neural network (QNN) to meet the above two requirements. Unlike traditional neural networks, features of the QNN are expressed by quaternion-valued vectors/matrix/tensors. Quaternion is a number system extended from the complex number, which include three imaginary parts and is expressed by $\boldsymbol{q} = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}$.

For privacy protection, we hide the input information into a random phase of quaternion-valued features (like the phase of a complex value) in the intermediate-layer of the QNN. The phase can be considered as the private key. Attackers without knowing the target phase cannot recover the input information from quaternion-valued features.

Fig. 1 shows the architecture of the QNN. It consists of an

[*]Equal contribution [1]Shanghai Jiao Tong University [2]Southern University of Science and Technology [3]Huawei International. Correspondence to: Quanshi Zhang <zqs1022@sjtu.edu.cn, John Hopcroft Center and MoE Key Lab of Artificial Intelligence AI Institute, Shanghai Jiao Tong University, China.>.

encoder, a processing module, and a decoder. The encoder extracts the real-valued feature, converts it into a quaternion-valued feature, and hides the input information in a target phase as the encryption process. The quaternion-valued feature is sent to the processing module for further process. The decoder decrypts the processed feature and computes the result. In order to enable the successful decryption of the decoder, the processing module has to ensure rotation equivariance. *I.e.* if the encoder rotates the target phase containing input information by a certain angle, the phases containing the input information in all layers of the processing module are rotated by the same angle. To achieve that, we need to revise layerwise operations in the neural network, including ReLU, batch-normalization, *etc.* to ensure the rotation equivariance property. However, by exhaustive searching, the attacker is still likely to recover the target phase in quaternion-valued features. Hence we further obfuscate the target phase by adversarial learning. *I.e.* the GAN generates quaternion-valued features to obfuscate the attacker.

The network revision can be broadly applied to DNNs with different architectures for various tasks. Experimental results showed that our QNNs outperformed other baselines in terms of privacy protection, yet the accuracy was not significantly affected.

Our contributions are summarized as follows: We propose a generic method to revise traditional networks into QNNs, which preserves input privacy by hiding sensitive information in a randomly chosen phase. Without knowing the phase, attackers can hardly infer any input information from features. Yet, anyone with the phase information can easily obtain correct outputs without much accuracy loss. Most importantly, QNNs incur far less computational overhead than crypto-based methods.

2. Related work

Complex-valued and quaternion-valued neural networks. Quaternions are a system extended from traditional complex numbers (Hamilton, 1848). (Reichert & Serre, 2014) applied complex-valued neurons to build richer, versatile representations in neural networks. (Danihelka et al., 2016) proposed a method to augment recurrent neural networks with associative memory based on complex-valued vectors, which achieved faster learning on multiple memorization tasks. According to (Trabelsi et al., 2018), using complex-valued parameters and complex-valued features in neural networks can achieve competitive performance with real-valued networks.

There are also several studies on quaternion-valued networks. (Zhu et al., 2018) represented color image and convolutional kernels in the quaternion domain to build quaternion convolutional neural networks, which outperformed real-valued neural networks in image classification and denoising tasks. (Parcollet et al., 2019) built quaternion recurrent neural networks and quaternion long-short term memory neural networks, and achieved a better performance than real-valued networks. (Kendall et al., 2015) localized the camera with the help of quaternions. In this paper, we convert intermediate-layer real-valued features of a traditional neural network into quaternion-valued features of a QNN, and hide the input information into a random phase for privacy protection. Unlike (Zhu et al., 2018; Parcollet et al., 2019), the QNN uses real-valued parameters, instead of quaternion-valued parameters.

Privacy protection in deep learning. Several studies have been proposed to protect privacy in deep learning. (Osia et al., 2017) used Siamese fine-tuning to reduce the level of sensitive information in the input, so that attackers cannot infer private properties using intermediate-layer features. (Li et al., 2017) proposed the PrivyNet to explore the trade-off between the privacy protection and the task accuracy. (Wang et al., 2018) applied a lightweight privacy protection mechanism consisting of data nullification and random noise addition. Homomorphic encryption is a cryptographic technique, which can be applied in the privacy protection in deep learning. (Zhang et al., 2016) used the BGV encryption scheme to encrypt data, and used the high-order backpropagation algorithm for training. (Mohassel & Zhang, 2017) distributed data among two non-colluding servers, where the model was trained using secure two-party computation. (Xiang et al., 2019) used complex-valued neural networks to protect privacy. Inspired by the homomorphic encryption, we use QNNs. Crucially, compared with the homomorphic encryption, QNNs will not boost the computational cost significantly.

3. Algorithm

3.1. Quaternion

Quaternion is a number system extended from the complex number. Unlike the complex number, a quaternion consists of three imaginary parts $q_1\boldsymbol{i}$, $q_2\boldsymbol{j}$, $q_3\boldsymbol{k}$, and one real part q_0, which is given as $\boldsymbol{q} = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}$. If the real part of a quaternion is zero ($q_0 = 0$), we call it a pure quaternion. The quaternion subject to $||\boldsymbol{q}|| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1$ is termed a unit quaternion. The products of basis elements $\boldsymbol{i}$, $\boldsymbol{j}$, $\boldsymbol{k}$ are given as $\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1$, and $\boldsymbol{ij} = \boldsymbol{k}$, $\boldsymbol{jk} = \boldsymbol{i}$, $\boldsymbol{ki} = \boldsymbol{j}$, $\boldsymbol{ji} = -\boldsymbol{k}$, $\boldsymbol{kj} = -\boldsymbol{i}$, $\boldsymbol{ik} = -\boldsymbol{j}$. Note that the multiplication of two imaginary parts is non-commutative, *i.e.* $\boldsymbol{ij} \neq \boldsymbol{ji}$, $\boldsymbol{jk} \neq \boldsymbol{kj}$, $\boldsymbol{ki} \neq \boldsymbol{ik}$. Each quaternion has a polar decomposition. The polar decomposition of a unit quaternion is defined as $e^{\boldsymbol{o}\frac{\theta}{2}} = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}(o_1\boldsymbol{i} + o_2\boldsymbol{j} + o_3\boldsymbol{k})$, *s.t.* $\sqrt{o_1^2 + o_2^2 + o_3^2} = 1$.

When we use a pure quaternion $\boldsymbol{q} = 0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}$ to represent a point $[q_1, q_2, q_3]^T \in \mathbb{R}^3$ in a 3D space, the rotation of the point around the axis $\boldsymbol{o} = o_1\boldsymbol{i}+o_2\boldsymbol{j}+o_3\boldsymbol{k}$, *s.t.* $\sqrt{o_1^2 + o_2^2 + o_3^2} = 1$, by the angle θ can be represented as $\boldsymbol{R}\boldsymbol{q}\overline{\boldsymbol{R}}$, where $\boldsymbol{R} = e^{\boldsymbol{o}\frac{\theta}{2}}$, and $\overline{\boldsymbol{R}} = e^{-\boldsymbol{o}\frac{\theta}{2}}$ is the conjugation of $\boldsymbol{R}$.

Given a pure quaternion-valued vector $\boldsymbol{x} = 0 + a\boldsymbol{i} + b\boldsymbol{j} + c\boldsymbol{k} \in \mathbb{H}^n$, and a real-valued vector $w \in \mathbb{R}^n$, we have

$$\boldsymbol{x}^T w = \sum_{v=1}^{n} \boldsymbol{x}_v w_v = 0 + (a^T w)\boldsymbol{i} + (b^T w)\boldsymbol{j} + (c^T w)\boldsymbol{k} \quad (1)$$

3.2. Overview of the QNN

In this paper, we revise a traditional DNN to construct a QNN. As Fig. 2 shows, we split the QNN into the following three modules:

- **Encoder module:** The encoder module is usually deployed at the local device. The encoder module extracts traditional real-valued features from the input, and encodes these real-valued features into quaternion-valued features, where each feature element is a quaternion. The encoder uses a specific phase of the quaternion-valued feature to encode the input information, *i.e.* the input information is hidden inside the target phase of each quaternion feature element. Information in other phases of the quaternion feature element represents noises to obfuscate attackers. We consider the target phase as a private key. Without the private key (the target phase), the attacker cannot invert the quaternion-valued back into the input. Then, the encoder module sends the encrypted quaternion-valued feature to the processing module. The target phase is kept by the local device. Note that parameters in the QNN are real-valued, instead of quaternion-valued.

- **Processing module:** The processing module is supposed to be deployed at the computation center, *e.g.* the cloud, where data can be processed efficiently. The processing module deals with the quaternion-valued feature received from the encoder module. All intermediate-layer features in the processing module are quaternion-valued. The processing module is not given the target phase, which encodes the input information. The task of privacy protection requires the layerwise processing of quaternion-valued features in the processing module to ensure the rotation equivariance property. *I.e.* the input information is hidden inside the same phase of all quaternion-valued features of all layers in the processing module. In this way, the processing module sends the output quaternion-valued feature to the decoder module.

- **Decoder module:** The decoder module is deployed at the local device. Because of the rotation equivariance of the processing module, the decoder module can use the target phase to decrypt the input information from the received quaternion-valued feature.

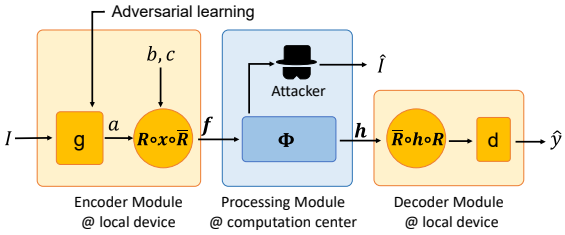

Figure 2. Architecture of the QNN. The QNN is divided into three modules: the encoder module, the processing module, and the decoder module. The encoder module extracts real-valued features from the input, and transform it into a quaternion-valued feature. The input information is hidden into a random phase of the quaternion-valued feature. The processing module deals with quaternion-valued features without knowing the target phase. The decoder module decrypts the input information with the target phase to get the final result.

3.3. Detailed design of the QNN

In this section, we introduce a set of basic rules to transform a traditional neural network into a QNN. We only revise the traditional real-valued feature to the quaternion-valued feature. But parameters in the QNN, *e.g.* weights in a filter, are still real numbers instead of quaternions.

Encoder: Given an input $I \in \mathbf{I}$, the encoder module g computes a traditional real-valued feature a as follows.

$$a = g(I) \in \mathbb{R}^n \quad (2)$$

Then the encoder module uses a and two fooling counterparts b, c to generate a quaternion-valued feature $\boldsymbol{x} = 0 + a\boldsymbol{i} + b\boldsymbol{j} + c\boldsymbol{k} \in \mathbb{H}^n$. Each element in $\boldsymbol{x}$ is a quaternion. Note that we can equivalently let $b = g(I)$ or $c = g(I)$ without loss of generality. We encrypt the quaternion-valued feature by rotating $\boldsymbol{x}$ along a random axis $\boldsymbol{o} = 0 + o_1\boldsymbol{i} + o_2\boldsymbol{j} + o_3\boldsymbol{k}$ by a random rotation angle θ, $o_1, o_2, o_3 \in \mathbb{R}$, $||\boldsymbol{o}|| = 1$, and obtain the encrypted feature $\boldsymbol{f} \in \mathbb{H}^n$ as follows.

$$\boldsymbol{f} = \Psi_{\boldsymbol{R}}(a) = \boldsymbol{R} \circ \boldsymbol{x} \circ \overline{\boldsymbol{R}} = \boldsymbol{R} \circ (0 + a\boldsymbol{i} + b\boldsymbol{j} + c\boldsymbol{k}) \circ \overline{\boldsymbol{R}} \quad (3)$$

where $\Psi_{\boldsymbol{R}}(\cdot)$ denotes the function which applies a random rotation $\boldsymbol{R}$ to the original quaternion-valued feature $\boldsymbol{x}$, $\boldsymbol{R} = e^{\boldsymbol{o}\frac{\theta}{2}} = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}(o_1\boldsymbol{i} + o_2\boldsymbol{j} + o_3\boldsymbol{k})$, and $\circ$ denotes the element-wise multipication. The encrypted feature $\boldsymbol{f}$ will be sent to the processing module Φ. In this way, we can consider $\boldsymbol{R}\boldsymbol{i}\overline{\boldsymbol{R}}$ as the target phase, which encodes the input information, and $\boldsymbol{R}$ can be taken as the private key.

Processing module: Inspired by homomorphic encryption, we revise the operation of each layer in the processing module to satisfy rotation equivariance of the quaternion-valued feature. The rotation equivariance property ensures that the input information is always encoded in the same phase

of all quaternion-valued features of all layers in the processing module. In this way, the decoder module can use the target phase to decrypt the input information from the quaternion-valued feature.

The rotation equivariance property can be summarized as follows. If we use $\boldsymbol{R} \circ \boldsymbol{x} \circ \overline{\boldsymbol{R}}$ to rotate quaternion-valued feature $\boldsymbol{x}$ along the axis $\boldsymbol{o}$ by the angle θ, then quaternion-valued feature elements in each intermediate layer of the processing module are supposed to be rotated along the same axis by the same angle, as follows.

$$\boldsymbol{\Phi}(\boldsymbol{R} \circ \boldsymbol{x} \circ \overline{\boldsymbol{R}}) = \boldsymbol{R} \circ \boldsymbol{\Phi}(\boldsymbol{x}) \circ \overline{\boldsymbol{R}} \tag{4}$$

Let us consider $\boldsymbol{h}_0 = \boldsymbol{\Phi}(\boldsymbol{x})$ without the rotation as the output of the processing module, *i.e.* $\theta = 0$, $\boldsymbol{R} = e^{\boldsymbol{o}\frac{\theta}{2}} = \mathbf{1}$. The input information is hidden in the imaginary part $\boldsymbol{i}$. Since all parameters in the processing module are real-valued, according to Equation (1), the output of the processing module can be represented in the form

$$\boldsymbol{h}_0 = \boldsymbol{\Phi}(\boldsymbol{x}) = 0 + (Aa)\boldsymbol{i} + (Ab)\boldsymbol{j} + (Ac)\boldsymbol{k}. \tag{5}$$

A is a real-valued matrix that represents effects that combine all non-linear transformations in $\boldsymbol{\Phi}(\boldsymbol{x})$, when $\boldsymbol{\Phi}(\boldsymbol{x})$ only uses ReLU as non-linear layers. In this way, the input information is still hidden in the imaginary part $\boldsymbol{i}$ of $\boldsymbol{h}_0$. Then, let us consider the rotation $\boldsymbol{R}$, $\boldsymbol{h} = \boldsymbol{\Phi}(\boldsymbol{R} \circ \boldsymbol{x} \circ \overline{\boldsymbol{R}})$. According to Equations (4) and (5), the output is given as

$$\begin{aligned} \boldsymbol{h} &= \boldsymbol{\Phi}(\boldsymbol{R} \circ \boldsymbol{x} \circ \overline{\boldsymbol{R}}) = \boldsymbol{R} \circ \boldsymbol{h}_0 \circ \overline{\boldsymbol{R}} \\ &= (\boldsymbol{R}\boldsymbol{i}\overline{\boldsymbol{R}}) \circ (Aa) + (\boldsymbol{R}\boldsymbol{j}\overline{\boldsymbol{R}}) \circ (Ab) + (\boldsymbol{R}\boldsymbol{k}\overline{\boldsymbol{R}}) \circ (Ac) \end{aligned} \tag{6}$$

In this way, the input information is hidden in the phase $\boldsymbol{R}\boldsymbol{i}\overline{\boldsymbol{R}}$ of $\boldsymbol{h}$. To ensure the above rotation equivariance, we recursively ensure rotation equivariance of the layerwise operation of each layer inside the processing module. The processing module can be represented as cascaded layers $\Phi(\boldsymbol{f}) = \Phi_L(\Phi_{L-1}(\cdots \Phi_1(\boldsymbol{f}))$, where $\Phi_l(\cdot)$ denotes the l-th layer in the processing model. Let $\boldsymbol{f}'$ denote the input feature of the l-th layer, then the layerwise operation is supposed to satisfy

$$\Phi_l(\boldsymbol{R} \circ \boldsymbol{f}' \circ \overline{\boldsymbol{R}}) = \boldsymbol{R} \circ \Phi_l(\boldsymbol{f}') \circ \overline{\boldsymbol{R}}. \tag{7}$$

Thus, this equation recursively ensures rotation equivariance in Equation (4).

In this paper, we revise the operation of each layer to satisfy both Equation (6) and (7). We consider six most widely-used layers, including the convolutional layer, the ReLU layer, the batch-normalization layer, the avg/max-pooling layer, the dropout layer, and the skip-connection operation.

Convolutional layer (or fully-connected layer): For the convolutional layer, we remove the bias term, and get $\text{Conv}(\mathbf{f}) = w \otimes \mathbf{f}$. Note that this revision can also be applied to the fully-connected layer, since the fully-connected layer can be considered as a special convolutional layer.

ReLU: We revise the ReLU layer as follows.

$$\text{ReLU}(\boldsymbol{f}_v) = \frac{\|\boldsymbol{f}_v\|}{\max\{\|\boldsymbol{f}_v\|, C\}} \cdot \boldsymbol{f}_v \tag{8}$$

where C denotes a positive scalar; $\boldsymbol{f}_v \in \mathbb{H}$ is the v-th element in the quaternion-valued feature $\boldsymbol{f}$.

Batch-normalization: We revise the batch-normalization operation as follows.

$$\text{norm}(\boldsymbol{f}_v^{(k)}) = \frac{\boldsymbol{f}_v^{(k)}}{\sqrt{\mathbb{E}_{k'}[\|\boldsymbol{f}_v^{(k')}\|^2] + \epsilon}} \tag{9}$$

where $\boldsymbol{f}_v^{(k)}$ denotes the v-th quaternion-valued element in the k-th sample in the batch. ϵ denotes a small positive scalar to prevent $\boldsymbol{f}_v^{(k)}$ from being divided by zero.

Avg/Max-pooling: Avg-pooling layers satisfy Equation (7) without any revisions. The max-pooling layer selects the quaternion feature element with the largest norm from the receptive field. We revise the max-pooling layer as follows.

$$\text{maxpool}(\boldsymbol{f}) = \boldsymbol{f} \circ m, \text{ where } \boldsymbol{f} \in \mathbb{H}^{n'}, m \in \{0, 1\}^{n'} \tag{10}$$

If the i-th element in $\boldsymbol{f}$ is selected (the selected quaternion feature element has the largest norm from the receptive field), then $m_i = 1$; otherwise, $m_i = 0$.

Dropout: We randomly dropout several quaternion-valued elements in the feature. The real part and three imaginary parts of dropped elements will all be set to zero.

Skip connection: The skip connection can be formulated as $\boldsymbol{f} + \Phi(\boldsymbol{f})$. If $\Phi(\boldsymbol{f})$ satisfies Equation (7), then the skip connection will also satisfy Equation (7).

All above revised operations satisfy Equation (7).

Decoder: Let $\boldsymbol{h} = \boldsymbol{\Phi}(\boldsymbol{f})$. Let d denote the decoder module, which can be implemented as a shallow network or a simple softmax layer. The decoder module can get the final result $\hat{y}$ as follows.

$$\hat{y} = d(\Psi_{\boldsymbol{R}}^{-1}(\boldsymbol{h})), \quad \Psi_{\boldsymbol{R}}^{-1}(\boldsymbol{h}) = \text{Im}_{\boldsymbol{i}}(\overline{\boldsymbol{R}} \circ \boldsymbol{h} \circ \boldsymbol{R}) \tag{11}$$

where $\Psi_{\boldsymbol{R}}^{-1}(\cdot)$ indicates the inverse function of $\Psi_{\boldsymbol{R}}(\cdot)$. The rotation in Equation (11) is the inverse of the rotation in Equation (3). $\text{Im}_{\boldsymbol{i}}(\cdot)$ denotes the operation that picks the $\boldsymbol{i}$ part from quaternions, and returns a real-valued feature.

3.4. Encoder based on GAN

To further boost the robustness to attacks, we need to adopt adversarial learning to learn an encoder. In adversarial learning, we use an encoder to generate quaternion-valued features to fool the attacker from obtaining the target phase. Therefore, we use a GAN to train the encoder.

The GAN (Goodfellow et al., 2014) includes a generator and a discriminator. In this paper, the generator can be viewed as the encoder module, while the discriminator can be viewed as the attacker. On the one hand, the generator generates quaternion-valued features that satisfy: 1. features contain enough information for the task; 2. features in the target phase and features in the other phase follow the same distribution, so that the discriminator cannot distinguish which phase contains the input information. On the other hand, the discriminator learns to distinguish the target phase that encodes input information.

Given an input $I \in \mathbf{I}$ and its real-valued feature $a = g(I)$ from the generator, we can encrpyt it with $\boldsymbol{R}$ as $\boldsymbol{f} = \Psi_{\boldsymbol{R}}(a) = \boldsymbol{R} \circ (0 + a\boldsymbol{i} + b\boldsymbol{j} + c\boldsymbol{k}) \circ \overline{\boldsymbol{R}}$, where b, c are fooling counterparts that do not contain input information. The attacker tries to estimate the most probable phase to decrypt $\boldsymbol{f}$, and get a. Let $\boldsymbol{R}'$ denote the rotation estimated by the attacker, and a' denote the decrypted feature, *i.e.* $a' = \Psi^{-1}_{\boldsymbol{R}'}(\boldsymbol{f}) = \text{Im}_{\boldsymbol{i}}(\overline{\boldsymbol{R}'} \circ \boldsymbol{f} \circ \boldsymbol{R}')$. a and a' are inputs of the discriminator D, and the discriminator needs to learn to distinguish a and a'. The generator g and the discriminator D are trained jointly. We adopt the WGAN (Arjovsky et al., 2017) to train the generator and the discriminator as follows.

$$\begin{aligned} \min_g \max_D L(g, D) &= \mathbb{E}_I[D(a) - \mathbb{E}_{\boldsymbol{R}' \neq \boldsymbol{R}}[D(a')]] \\ &= \mathbb{E}_I[D(a) - \mathbb{E}_{\boldsymbol{R}' \neq \boldsymbol{R}}[D(\Psi^{-1}_{\boldsymbol{R}'}(\Psi_{\boldsymbol{R}}(a)))] \end{aligned} \tag{12}$$

To jointly optimize the GAN loss and the task loss, the overall loss is written as follows.

$$\min_{g,\Phi,d} \max_D Loss = \min_{g,\Phi,d}[\max_D L(g, D) + L_{\text{task}}(\hat{y}, y)] \tag{13}$$

where L_{task} is the loss for the task, and y is the ground-truth label. As for fooling parts b and c, we compute them as $b = g(I')$ and $c = g(I'')$ to simplify the implementation, where I' and I'' denote images different from I.

3.5. Attackers to QNNs

We design two types of attackers to test the performance of the privacy protection of QNNs.

Feature inversion attackers: Let the attacker retrieve quaternion-valued features in the processing module. The feature inversion attacker usually trains another neural network to reconstruct the input. In this paper, we implement two versions of feature inversion attackers.

- The first inversion attacker aims to estimate the target phase of the quaternion-valued feature, which encodes the input information, *i.e.* to estimate $\boldsymbol{R}'$. Then the attacker reconstructs the input using the feature decrypted with $\boldsymbol{R}'$, $a' = \Psi^{-1}_{\boldsymbol{R}'}(\boldsymbol{f})$. The attacker learns another neural network to reconstruct the input, *i.e.* $\hat{I} = \text{dec}_1(a')$.

Thus, the core technique is to enumerate all phases, and train a classifier D' to judge whether the decrypted feature a' encodes the input information. The classifier D' is similar to the discriminator D in the GAN.

- The second feature inversion attacker directly uses the encrypted quaternion-valued feature to reconstruct the input. The attacker is trained and tested with $\boldsymbol{f} = \Psi_{\boldsymbol{R}}(a)$, *i.e.* $\hat{I} = \text{dec}_2(\boldsymbol{f}) = \text{dec}_2(\Psi_{\boldsymbol{R}}(a))$.

Property inference attackers: According to (Ganju et al., 2018), the "property" refers to the input attribute. We design the first property inference attacker to use the intermediate-layer feature to infer sensitive attributes of inputs. For example, if we train a neural network for the age recognition task, then the property such as gender and race should also be hidden in intermediate-layer features. In this paper, we test four versions of property inference attackers.

- The first property inference attacker trains a classifier, which uses the quaternion-valued feature to infer sensitive attributes of the input. Specifically, the attacker decrypts $\boldsymbol{f}$ using an estimated $\boldsymbol{R}'$ as $a' = \Psi^{-1}_{\boldsymbol{R}'}(\boldsymbol{f})$. The classifier D' is used to judge whether the decrypted feature a' encodes the input information. The attacker learns another classifier to estimate $attr = \text{net}(I)$. The attacker trains the classifier using pairs of inputs and their attributes, $(I, attr)$, and tests the classifier using the reconstructed input $\hat{I}$.

The property inference attacker is supposed to deal with quaternion-valued features of any layer in the processing module. However, all intermediate-layer features are computed based on the input feature $\boldsymbol{f}$, which contains the richest information of the input. To simplify the implementation, we only consider hacking the feature $\boldsymbol{f}$ in this paper. This is uniformly applied to the 2nd, 3rd, and 4th attackers.

- The second property inference attacker is similar to the first property inference attacker. The only difference is that the attacker trains the classifier using pairs of the decrypted feature a' and ground-truth attributes, $(a', attr)$. *I.e.* the attacker learns the classifier $attr = \text{net}(a')$, and is tested using the obtained a' during the attack process.

- The third property inference attacker is similar to the first property inference attacker. The attacker learns the classifier using annotations of reconstructed inputs and attributes, $(\hat{I}, attr)$, *i.e.* $attr = \text{net}(\hat{I})$. The classifier is tested using the reconstructed inputs $\hat{I}$.

- The forth property inference attacker applies the k-nearest neighbors (k-NNs) instead of a neural network to infer sensitive attributes. For images whose decrypted features a' are close to each other in the feature space, we consider these images have similar attributes. In this way, the attacker uses the k-nearest neighbors in the feature space to infer the sensitive attributes.

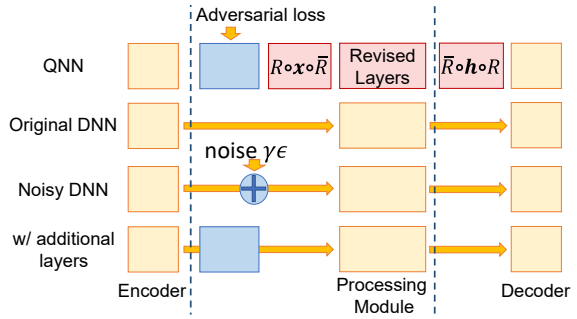

Figure 3. Alignment of architectures between neural networks.

3.6. K-anonymity

Interestingly, we can analyze the performance of the QNN from the perspective of the k-anonymity. *K-anonymity* means that when attackers try to reconstruct an input from the quaternion-valued feature, they can get at least k different results, while only one of them is the correct input. We visualize the decrypted result $\hat{I}$ obtained from the feature inversion attacker or the property inference attacker for analysis. Fig. 5 shows reconstructed inputs from different phases with similar confidence estimated by the attacker. Table 6 shows that the QNN can roughly achieve the k-anonymity.

4. Experiments

In this section, we conducted a series of experiments to test QNNs. We converted a number of classic neural networks into QNNs. Theoretically, the QNN can be applied to different tasks. In this paper, we tested QNNs on tasks include object classification and face attribute estimation, considering both the feature inversion attacker and the property inference attacker.

We trained QNNs using the CIFAR-10, CIFAR-100 datasets (Krizhevsky & Hinton, 2009), which contained small images, for object classification. Besides, we used the CUB200-2011 dataset (Wah et al., 2011) and the CelebA dataset (Liu et al., 2015) for object classification and face attributes estimation with large image. We revised classic neural networks into QNNs, which included LeNet (LeCun et al., 1998), residual networks (He et al., 2016), VGG-16 (Simonyan & Zisserman, 2015), and AlexNet (Krizhevsky et al., 2012).

Network architectures: Fig. 3 compares architectures between the original neural network and its corresponding QNN. The encoder/processing/decoder modules of each original neural network are introduced as follows. For the LeNet, the encoder consisted of all layers before the second convolutional layer, and there was only a softmax-layer in the decoder. For the residual network, the encoder consisted of all layers before the first 16×16 feature map. The decoder consisted of all layers after the first 8×8 feature map. For the AlexNet, the encoder consisted of the first convolutional layers, and the decoder consisted of fully-connected layers and the softmax layer. For the VGG-16, all layers before the last 56×56 feature map compromised the encoder, and the decoder consisted of fully-connected layers and the softmax layer. Note that for each encoder, we add the GAN at the end of the encoder.

Baselines: We proposed four baselines for comparison. As shown in the second row of Fig. 3, the original network without any revision was used as the first baseline which was denoted as *Original DNN*. We divided the original network into encoder, processing module, and decoder in the same way as QNN. The third row of Fig. 3 shows the second baseline. Noise addition could also be used for privacy protection. We added noises to the output a of the encoder, *i.e.* $a + \gamma\epsilon$, where ϵ denoted a random noise vector, and γ was a scalar. The second baseline was denoted as *Noisy DNN*, and we trained *Noisy DNN* with $\gamma = 0.2, 0.5, 1.0$. The third baseline is shown in the last row of Fig. 3, which was denoted as *w/ additional layers*. Since the insertion of the GAN increased the layer number of the QNN. For a fair comparison, we also added the GAN architecture into the baseline network, but the baseline network was learned without the GAN loss. The forth baseline was the (Xiang et al., 2019), which was constructed using the same division of modules of the QNN. For simplicity, the forth baseline was denoted as *Complex NN*. Theoretically speaking, QNNs had more potential phases to hide the input information than Complex NNs, and used more fooling counterparts to obfuscate attackers. Besides, the rotation in 3D space was more complicated than rotation in 2D space. Therefore, the QNNs were supposed to have a better privacy protection performance than Complex NNs.

Attack: We applied two kinds of attackers in Sec. 3.5.

• *Inversion Attack:* The inversion attacker was implemented based on U-Net (Ronneberger et al., 2015).We revised the U-Net to construct the attacker for inversion attacks. The intermediate-layer feature was upsampled to the size of the input, which was fed into the inversion model. There were four down-sample blocks and four up-sample blocks. Each block had six convolutional layers for better performance. The output of the inversion model had the same size as the input. We used intermediate-layer features from QNNs as the input of the attacker. In this way, we used the U-Net to construct both the first inversion attacker and the second inversion attacker, according to Sec. 3.5. The first inversion attacker trained the attacker with real-valued features, and tested the attacker with the output of D. The second inversion attack trained and tested the attacker with the encrypted quaternion-valued features. To mimic the procedure of hacking the privacy, we randomly sampled the rotation axis and rotation angle for 1000 times. The sample with the

Table 1. Classification error rates and reconstruction errors indicating the capacity of privacy protection.

Classification Error Rate(%)						
Model	Dataset	Original DNN	w/ additional layers	Complex NN	QNN
ResNet-20	CIFAR-10	11.56	9.68	10.91	**9.21**
ResNet-32	CIFAR-10	11.13	9.67	10.48	**9.82**
ResNet-44	CIFAR-10	10.67	9.43	11.08	**9.54**
ResNet-56	CIFAR-10	10.17	9.16	11.53	**9.24**
ResNet-110	CIFAR-10	10.19	9.14	11.97	**9.31**

Reconstruction Errors						
Original DNN	w/ additional layers	Complex dec(a')	Complex dec(x)	Quaternion dec(a')	Quaternion dec(x)
0.0906	0.1225	0.2664	0.2420	**0.3014**	0.2702
0.0930	0.1171	0.2569	0.2412	**0.2813**	0.2412
0.0933	0.1109	0.2746	0.2419	**0.3123**	0.2421
0.0989	0.1304	0.2804	0.2377	**0.3083**	0.2403
0.0896	0.1079	**0.3081**	0.2495	0.3028	0.2379

Reconstruction Errors										
Model	Dataset	Original DNN	w/additional layers	Noisy DNN $\gamma = 0.2$	Noisy DNN $\gamma = 0.5$	Noisy DNN $\gamma = 1.0$	Complex dec(a')	Complex dec(x)	Quaternion dec(a')	Quaternion dec(x)
LeNet	CIFAR-10	0.0769	0.1208	0.0948	0.1076	0.1274	0.2405	0.2353	**0.2877**	0.2303
LeNet	CIFAR-100	0.0708	0.1314	0.0950	0.1012	0.1286	0.2700	0.2483	**0.2996**	0.2528
ResNet-56	CIFAR-100	0.0929	0.1029	0.1461	0.1691	0.2017	0.2593	0.2473	**0.3057**	0.2592
ResNet-110	CIFAR-100	0.1050	0.1092	0.1483	0.1690	0.2116	0.2602	0.2419	**0.3019**	0.2543
VGG-16	CUB200-2011	0.1285	0.1202	0.1764	0.0972	0.1990	0.2803	0.2100	**0.3133**	0.1945
AlexNet	CelebA	0.0687	0.1068	-	-	-	**0.3272**	0.2597	0.3239	0.2657

Figure 4. Reconstructed examples. Left: images from the CelebA dataset; right: images from the CIFAR-10 dataset.

Table 2. Classification error rates on variety of models and datasets.

Model	Dataset	Original DNN	w/ additional layers	Noisy DNN $\gamma = 0.2$	Noisy DNN $\gamma = 0.5$	Noisy DNN $\gamma = 1.0$	Complex NN	QNN
LeNet	CIFAR-10	19.78	21.52	24.15	27.53	34.43	17.95	**11.45**
LeNet	CIFAR-100	51.45	49.85	56.65	67.66	78.82	49.76	**37.78**
ResNet-56	CIFAR-100	53.26	44.38	57.24	61.31	74.17	**44.37**	44.86
ResNet-110	CIFAR-100	50.64	44.93	55.19	61.12	71.31	50.94	**42.05**
VGG-16	CUB200-2011	56.78	63.47	69.20	99.48	99.48	78.50	**70.86**
AlexNet	CelebA	14.17	9.49	-	-	-	15.94	**8.80**

highest output of the discriminator was considered as the optimal feature to reconstruct the original image.

• *Inference attacker:* For the inference attack, we used the CelebA and CIFAR-100 datasets for testing. For the CelebA dataset, we selected 10 attributes as private attributes. We revised an AlexNet to a QNN, and trained the QNN to estimate other 30 attributes, whereas the attacker based on ResNet-50 used the intermediate-layer feature to estimate private attributes. For the CIFAR-100 dataset, we trained a quaternion-valued ResNet-56 to classify major 20 superclasses of CIFAR-100. The attacker was constructed based on ResNet-56, and used the intermediate-layer feature to infer 100 minor classes, which were considered as sensitive information in this experiment.

Evaluation metrics of privacy protection: We used four metrics to evaluate the performance of privacy protection in terms of inversion attackers: (1) the pixel-level reconstruction error $\mathbb{E}[\|\hat{I} - I\|]$, where the pixel value was scaled to $[0, 1]$. (2) The average difference angle $\Delta\theta$ between $\mathbf{x}^{R^*}$ and $\mathbf{x}^{\hat{R}}$, where $\mathbf{x}^{R^*} = \boldsymbol{R^* i \overline{R}^*}$ and $\mathbf{x}^{\hat{R}}$ were unit quaternion vectors of $\mathbf{x}$-axis after rotation using R^* and $\hat{R}$, respectively. (3) The rank of the estimated sample. Let us recover two samples using two similar phases. According to our experience, when the angle between phases was less than $\Delta\theta = 5^\circ$, the two samples usually presented the same object entity. Thus, we estimated the number of object entities that were more similar to the input than the recovered sample. For the QNN, the rank of the estimated sample is computed as $\frac{2}{1-\cos\Delta\theta}$. For the Complex NN, the rank of the estimated

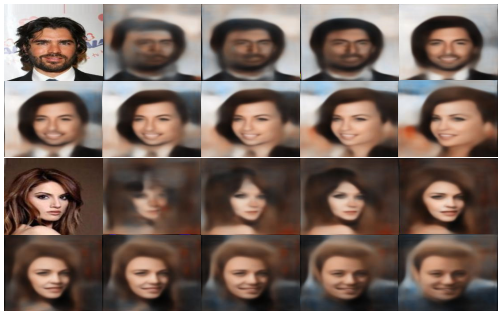

Figure 5. CelebA images reconstructed using different phases. The first image of every two rows is the input image. We pick the most meaningful images.

Table 3. Failure rate of identifying the reconstructed image by human annotators.

Model	Dataset	Original DNN	w/ additional layers	Noisy DNN $\gamma = 0.2$	Noisy DNN $\gamma = 0.5$	Noisy DNN $\gamma = 1.0$	Complex dec(a')	Complex dec(x)	Quaternion dec(a')	Quaternion dec(x)
LeNet	CIFAR-10	0.16	0.12	0.20	0.20	0.24	0.82	0.92	0.90	**0.96**
LeNet	CIFAR-100	0.16	0.12	0.20	0.64	0.72	0.80	0.92	0.94	**1.00**
ResNet-56	CIFAR-100	0.06	0.06	0.08	0.10	0.36	0.72	0.88	0.90	**1.00**
ResNEt-110	CIFAR-100	0.04	0.12	0.10	0.16	0.36	0.80	0.86	0.94	**0.98**
VGG-16	CUB200-2011	0.06	0.06	0.08	0.02	0.14	0.86	0.84	**0.86**	0.74
AlexNet	CelebA	0.04	0.24	-	-	-	0.96	1.00	0.84	**1.00**

Table 4. Average error of the estimation angle and the failure rate. Complex error and quaternion error represented the average error of estimated phase on Complex NN and QNN, respectively.

		Average Error	
Model	Dataset	Complex Error	Quaternion Error
ResNet-20	CIFAR-10	0.7890±0.3722	1.4803±0.7004
ResNet-32	CIFAR-10	0.7820±0.3630	1.3322±0.6785
ResNet-44	CIFAR-10	0.8411±0.6200	1.4610±0.6905
ResNet-56	CIFAR-10	0.8088±0.5848	1.4733±0.6932
ResNet-110	CIFAR-10	0.8048±0.4535	1.4461±0.6955
LeNet	CIFAR-10	0.7884±0.4147	1.3511±0.6765
LeNet	CIFAR-100	0.8046±0.5279	1.3842±0.6694
ResNet-56	CIFAR-100	0.7898±0.5544	1.3837±0.7010
ResNet-110	CIFAR-100	0.7878±0.3775	1.3515±0.6558
VGG-16	CUB200	1.5572±0.8778	1.3589±0.7120
AlexNet	CelebA	0.8500±0.5811	1.3833±0.6767

Table 5. Experimental results of inference attackers. net(I), net(a'), net($\hat{I}$), k-NN represented the first, the second, the third, and the forth inference attacker, respectively.

		Classification Error Rate	net(I)	net(a')	net($\hat{I}$)	k-NN		
	Structure					$k=1$	$k=3$	$k=5$
CIFAR-100	w/ additional layers	18.73	68.72	38.50	40.28	73.38	68.37	71.16
	Complex NN	26.77	94.53	87.17	89.56	94.44	93.63	92.50
	QNN	21.60	**96.73**	**94.22**	**95.25**	**98.19**	**97.60**	**97.42**
CelebA	w/ additional layers	8.04	19.14	13.17	14.01	20.14	17.26	16.20
	Complex NN	14.75	**25.72**	**22.21**	22.61	31.69	27.90	26.41
	QNN	8.20	25.03	22.19	**23.26**	**32.77**	**28.81**	**27.42**

sample is computed as $\frac{2\pi}{\Delta\theta}$. (4) The reconstruction failure rate of human identification, *i.e.* we used human annotators to judge whether they can identify the input based on the reconstructed sample $\hat{I}$. For the privacy performance of inference attacks, we could use the accuracy of attackers as the evaluation metric. What's more, we can compute the processing speed of different models to measure the efficiency of QNNs.

Experimental Results and Analysis: Table 1 and Table 2 show the performance of QNNs and baselines. Complex $dec(a')$ and Complex $dec(x)$ represented the first inversion attacker and the second inversion attacker on Complex NN, respectively. Quaternion $dec(a')$ and Quaternion $dec(x)$ represented the first inversion attacker and the second inversion attacker on QNN, respectively. Classification error rate results showed that QNNs achieved a better performance than Complex NNs. Compared with Original DNN and DNNs with additional layers, the accuracy of QNNs was not significantly affected. As for the reconstruction error, a higher value indicated a better privacy protection performance. The reconstruction error of QNNs was higher than other networks, *i.e.* QNNs exhibited better performance of privacy protection than other networks.

Table 6. Rank of the estimated sample (first row), and time cost of inference (second row).

	w/ additional layers	Complex NN	QNN	Homomorphic encryption
Rank	-	36	525.6	-
Time cost (s/image)	0.0004	0.0007	0.0011	3.56

Fig. 4 visualizes several examples from inversion attackers. We only provided results from partial experiments constrained by the space. Table 4 shows averages and standard errors of $\Delta\theta$. A smaller value of the average of $\Delta\theta$ indicated that attackers were easier to estimate the target phase. Table 3 shows the subjective failure rate according to the judgement of humans. Attackers could not fetch the input information from encrypted features. It was more difficult to estimate the target phase of quaternion-valued features than complex-valued features.

Table 5 shows the result of inference attackers. A higher value of inference error indicated a better privacy protection performance. Attackers on QNNs had a higher inference classification error than attackers on DNNs with additional layers, and attackers on Complex NN. Thus, QNNs protected private attributes from attackers more effectively.

Fig. 5 shows images reconstructed using different phases. Table 6 shows the rank of the estimated sample. A higher rank value indicated better performance of privacy protection. The rank of the QNN was higher than the Complex NN. *I.e.* it was more difficult for attackers on QNNs to find the target phase than attackers on Complex NNs.

Table 6 also shows the time cost of the inference process of DNNs. The time cost of homomorphic encryption was from the framework Gazelle (Juvekar et al., 2018), which trained a small network with 3 fully-connected layers from (Mohassel & Zhang, 2017) using the CIFAR-10 dataset. For other networks, we used networks revised from the ResNet-56, which were deeper than the network used by Gazelle. However, the inference time cost of the QNN was much less than

Gazelle, and was comparable with traditional DNNs.

5. Conclusion

In this paper, we propose a method to protect the privacy of inputs. Our method transforms traditional neural networks into QNNs, which use quaternion-valued features as intermediate-layer features. The input information is hidden in a random phase of quaternion-valued features. Experiments showed the effectiveness of the privacy protection of QNNs. The QNN has much lower computational cost than the homomorphic encryption.

References

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein generative adversrial networks. *In Proceedings of the 34th International Conference on Machine Learning (ICML)*, pp. 214–223, 2017.

Danihelka, I., Wayne, G., Uria, B., Kalchbrenner, N., and Graves, A. Associative long short-term memory. *In Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

Dosovitskiy, A. and Brox, T. Inverting visual representations with convolutional networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4829–4837, 2016.

Ganju, K., Wang, Q., Yang, W., Gunter, C. A., and Borisov, N. Property inference attacks on fully connected neural networks using permutation invariant representations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 619–633, 2018.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. *In NIPS*, 2014.

Hamilton, W. R. On quaternions; or on a new system of imaginaries in algebra. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, 33(219):58–60, 1848.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *In CVPR*, 2016.

Juvekar, C., Vaikuntanathan, V., and Chandrakasan, A. P. Gazelle: A low latency framework for secure neural network inference. *In arXiv: 1801:05507*, 2018.

Kendall, A., Grimes, M., and Cipolla, R. Posnet: A convolutional network for real-time 6-dof camera relocalization. *In Proceedings of the IEEE international conference on computer vision (ICCV)*, 2015.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. *In Computer Science Department, University of Toronto, Tech. Rep*, 1, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *In NIPS*, 2012.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *In Proceedings of the IEEE*, 1998.

Li, M., Lai, L., Suda, N., Chandra, V., and Z.Pan, D. Privynet: A flexible framework for privacy-preserving deep neural network training with a fine-grained privacy control. *In arXiv preprint arXiv:1709.06161*, 2017.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Mahendran, A. and Vedaldi, A. Understanding deep image representations by inverting them. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5188–5196, 2015.

Melis, L., Song, C., De Cristofaro, E., and Shmatikov, V. Exploiting unintended feature leakage in collaborative learning. *IEEE*, 2018.

Mohassel, P. and Zhang, Y. Secureml: A system for scalable privacy-preserving machine learning. *In 2017 38th IEEE Symposium on Security and Privacy (SP)*, pp. 19–38, 2017.

Osia, S. A., Shamsabadi, A. S., Sajadmanesh, S., Taheri, A., Katevas, K., R.Rabiee, H., D.Lane, N., and Haddadi, H. A hybird deep learning architecture for privacy-preserving mobile analytics. *In arXiv preprint arXiv:1703.02952*, 2017.

Parcollet, T., Ravanelli, M., Morchid, M., Linares, G., Trabelsi, C., Mori, R. D., and Bengio, Y. Quaternion recurrent neural networks. *In International Conference on Learning Representations (ICLR)*, 2019.

Reichert, D. P. and Serre, T. Neuronal synchrony in complex-valued deep networks. *In International Conference on Learning Representations (ICLR)*, 2014.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 234–241, 2015.

Shokri, R., Stronati, M., Song, C., and Shmatikov, V. Membership inference attacks against machine learning models. *IEEE Symposium on Security and Privacy (SP)*, pp. 3–18, 2017.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *In ICLR*, 2015.

Trabelsi, C., Bilaniuk, O., Zhang, Y., Serdyuk, D., Subramanian, S., Santos, J. F., Mehri, S., Rostamzadeh, N., Bengio, Y., and Pal, C. J. Deep complex networks. *In International Conference on Learning Representations (ICLR)*, 2018.

Wah, C., Branson, S., Welinder, P., Perona, P., and Belongie, S. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

Wang, J., Zhang, J., Bao, W., Zhu, X., Cao, B., and S.Yu, P. Not just privacy: Improving performance of private deep learning in mobile cloud. *In KDD*, 2018.

Xiang, L., Zhang, H., Ma, H., Zhang, Y., Ren, J., and Zhang, Q. Complex-valued neural networks for privacy protection. In *arXiv preprint arXiv:1901.09546*, 2019.

Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. *European Conference on Computer Vision (ECCV)*, pp. 818–833, 2014.

Zhang, Q., T.Yang, L., and Chen, Z. Privacy preserving deep computation model on cloud for big data feature learning. *IEEE Transactions on Computers*, 65(5):1351–1362, 2016.

Zhang, Z., Chang, E.-C., and Liang, Z. Adversarial neural network inversion via auxiliary knowledge alignment. *Proceedings of the Computer and Communications Security (CCS)*, pp. 225–240, 2019.

Zhu, X., Xu, Y., Xu, H., and Chen, C. Quaternion convolutional neural networks. *In Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 631–647, 2018.