Análise de Amostras Sintéticas de Sinais de Sonar Passivo Geradas por Redes Neurais Generativas Adversariais

Júlio de C. V. Fernandes
Universidade Federal do Rio de Janeiro
Email: juliocvf@poli.ufrj.br

Natanael N. M. Junior
Universidade Federal do Rio de Janeiro
Email: natmourajr@lps.ufrj.br

José Manoel de Seixas
Universidade Federal do Rio de Janeiro
Email: seixas@lps.ufrj.br

***Abstract*—In naval warfare, several techniques have been developed for the detection and classification of war vessels. Given the confidential nature of the data it is extremely difficult to get a hold of large quantities of data which makes it extremely hard to use techniques that rely on abundant data, such as deep learning. This paper proposes the use of generative adversarial neural networks for the generation of synthetic samples that can later be used in training of classifiers. This paper focuses on the generation process and the qualifying of such samples.**

***Keywords*—Sonar Systems, Neural Networks, Generative Adversarial Neural Networks (GAN), Deep Learning.**

I. Introdução

Um sistema **SO**und **NA**vigation **R**anging, ou, como é mais comumente conhecido, um sistema sonar, é um conjunto de técnicas que utiliza o som que se propaga através da água para comunicação, navegação e detecção de objetos [1]. Desde sua invenção, os sonares encontraram um vasto número de aplicações tanto para usos civis como, por exemplo, reconhecimento de marcos subaquáticos, rastreamento e estudo de animais, detecção de minas, aplicações em petróleo e gás, como também militares. Os submarinos dependem, consideravelmente, dos sistemas de sonares para navegar e adquirir informações sobre as condições e profundidade do mar, bem como para a detecção e classificação de navios [2], chamados neste contexto de contatos.

Existem dois tipos de sistemas sonar usados em navios: o ativo e o passivo [16]. No sistema de sonar ativo, o navio envia um sinal e utiliza seus ecos para detectar, classificar e localizar um possível alvo [3]. Enquanto isso, em um sistema de sonar passivo não há sinal emitido obrigando o sonar a trabalhar analisando os sons emitidos pelos possíveis alvos para a detecção e classificação destes [3]. Este trabalho foi desenvolvido dentro do contexto do sonar passivo, de uso militar. Deste modo, sinais, de sonares passivos são os ruídos emitidos pelas diversas fontes sonoras presentes no ambiente subaquático, sendo que, para este trabalho, os sinais de interesse são aqueles provenientes de navios, enquanto que o ruído de fundo é composto por todos os outros ruídos do ambiente subaquático, que podem incluir os ruídos emitidos pela vida marinha, bem como ruídos emitidos por plataformas de petróleo, por exemplo.

Dada a grande quantidade de ruídos de fundo presente no ambiente marítimo, a classificação de alvos se torna um problema de alta complexidade, pois a assinatura de um sinal de interesse pode ser mascarada pelo ruído de fundo. Os efeitos do ruído de fundo tem forte impacto no problema de classificação de contatos pois diversas fontes sonoras ocupam a mesma banda de frequência. Não obstante, a quantidade de dados disponíveis para o treinamento de um modelo pode ser bastante diminuta, especialmente em se tratando de dados com possíveis usos militares.

Nos últimos anos técnicas de aprendizado profundo vêm recebendo grande atenção [6] devido aos resultados que as diversas topologias de redes neurais profundas vem obtendo nos mais variados campos de pesquisa, muitas das vezes esses resultados são melhores do que o estado da arte no campo [6]. De forma geral podemos pensar no aprendizado profundo como um ramo da área de aprendizado de máquina que consegue, através do uso de um grande número de parâmetros (que para serem ajustados requerem um grande número de dados), de forma automática representações de alto nível dos dados [5] sendo que cada camada constrói uma representação de mais alto nível sobre a representação anterior [7]. Neste artigo exploramos técnicas de aprendizado profundo para a geração de amostras sintéticas e a qualificação destas amostras. Técnicas de aprendizado profundo já foram utilizadas para a geração de amostras sintéticas com os mais diversos propósitos como em: [21] aonde foram utilizadas para detecção de fraude em cartões de crédito, [23] para a detecção de anomalias em imagens médicas, [24] [26] para a geração de imagens de maior resolução, [22] denoising e [25] para o treinamento de classificadores.

O artigo está organizado da seguinte forma: na Seção II, o processamento básico de um sinal de sonar passivo é descrito. As redes neurais adversariais são brevemente expostas na Seção III. O método desenvolvido é explicado na Seção IV. Os resultados experimentais são analisados na Seção V, e as conclusões são derivadas na Seção VI.

II. Sistema de Sonar Passivo

A cadeia de processamento, do ruído irradiado por navios, de um sistema de sonar passivo consiste na aquisição do ruído, na conformação dos feixes, na determinação da direção de

Fig. 1. Diagrama de blocos da análise LOFAR.

chegada (determinação da direção do objeto que emitiu o sinal sendo analisado), na análise dos sinais (que pode ser feita tanto no domínio do tempo quanto no domínio da frequência), na classificação destes sinais e, se estes forem de interesse, no seu acompanhamento.

A caracterização destes sinais no domínio da frequência pode ser feite, dentre outras maneiras, pelas análises LOFAR (LOw Frequency Analysis and Recording) [8] quanto pela análise DEMON (Demodulation of Envelope Modulation On Noise) [20]. Este trabalho foi desenvolvido utilizando-se a análise LOFAR.

A. LOFAR

A análise LOFAR é uma análise de banda larga que trabalha sobre o ruído de máquina de um dado navio. Podemos ver o fluxograma da análise LOFAR na Fig. 1. De maneira resumida, esta análise consiste dos seguintes passos: primeiramente, o sinal é decimado e, em seguida, multiplicado por uma janela Hanning para enfatizar a faixa de frequência de interesse. Depois, a representação de frequência do sinal é obtida aplicando-se a transformada de Fourier (STFT - do inglês, *Short Time Fourier Transform*). Na sequência, o algoritmo TPSW (do inglês, *Two Pass Split Window*) [9] é utilizado para atenuar o ruído de fundo e normalizar o sinal. O resultado dessa análise pode ser exibido em um lofargrama, como visto na Fig. 2 no qual, a informação de frequência processada de cada janela é exibida em cada linha do eixo x enquanto o eixo y representa o tempo. Neste artigo imagens são criadas a partir da análise LOFAR. Esta forma de processar o lofargrama foi motivada pelos bons resultados obtidos usando-se redes neurais, especialmente a *Convolutional Neural Networks* (CNN) [5], com processamento de imagens. A escolha por tratar o lofargrama como uma imagem, ao invés de processar cada linha deste individualmente se dá pelo fato de uma imagem (combinando várias janelas) conter mais informação do que apenas um espectro, além da habilidade de tratar as imagens de maneira eficiente com redes neurais.

III. Redes Neurais Generativas Adversariais

Em [11], Goodfellow et al. introduziram o conceito de redes neurais adversariais generativas (do inglês, Generative Adversarial Neural Networks - GAN), que constituem uma classe de modelos não supervisionados que tentam gerar dados sintéticos com a mesma distribuição de probabilidade dos dados de treinamento. Goodfellow propôs um novo paradigma de treinamento em que duas redes neurais[1], chamadas de gerador e discriminador, participariam de um jogo de soma zero de dois jogadores, segundo a definição constando na teoria

[1]A primeira formulação de GAN utilizava redes neurais feedforward mas logo o conceito de GAN foi aplicado com diversas outras topologias.

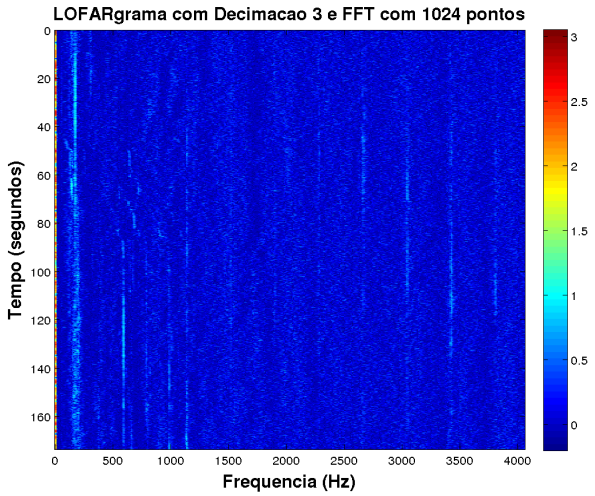

Fig. 2. Lofargrama. O espectro da janela sendo processada é representado no eixo x enquanto que o tempo é representado no eixo y.

de jogos [10], que permite às redes aprender a distribuição dos dados. A estrutura geral do jogo é bastante simples. O gerador gera amostras sintéticas e o discriminador têm como objetivo diferenciar amostras reais destas amostras sintéticas. O objetivo do gerador é então gerar amostras cada vez mais reais de modo que o discriminador não consiga diferenciar entre as amostras reais e as amostras sintéticas. Enquanto isso, a tarefa do modelo discriminador é justamente discriminar se uma dada amostra é real ou não sendo assim, ao longo do jogo, o discriminador apreende cada vez mais identificar uma amostra real e força o gerador a gerar amostras sintéticas cada vez mais verossímeis. A estrutura geral de uma GAN pode ser vista na Fig. 3.

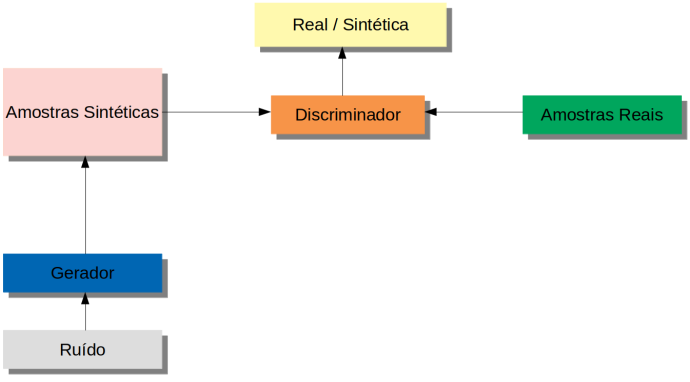

Fig. 3. Visão geral da estrutura da GAN, que consiste em dois modelos, o Gerador G e o Discriminador D. G recebe ruído como entrada e gera uma amostra de dados. O modelo D é treinado com amostras do conjunto de treinamento e amostras geradas por G. Ele aprende como separar amostras reais de amostras geradas e é treinado para produzir 1 para o primeiro e 0 para o segundo. G é treinado como qualquer outra rede por backpropagation [5] vindo de D. Durante o treinamento, G gera amostras e D as julga. O objetivo aqui é usar as amostras que foram realistas o suficiente para enganar D para melhorar G. É importante notar que G nunca é mostrado qualquer amostra real sendo treinado apenas através do *feedback* que recebe de D.

O treinamento destas redes ocorre simultaneamente e pode ser matematicamente descrito como: Defina p_d sobre o espaço X como a verdadeira distribuição dos dados de treinamento (isto é, os dados que queremos modelar), defina p_z sobre o espaço Z como o ruído de entrada do gerador (normalmente o ruído segue a distribuição uniforme ou normal) e, por fim, defina G(.;θ_g) e D(.;θ_d) como as parametrizações por θ_g e θ_d do gerador e do discriminador, respectivamente. Deste modo o objetivo de G é definir uma função que mapeie os espaços Z e X, $z \to x$ onde $x \in X$ e $z \in Z$. Definindo p_G como a distribuição de probabilidade obtida através da amostragem de G(z) podemos definir a seguinte função custo:

$$\min_{\theta_g} \max_{\theta_d} V(D,G) = E[logD(x)] + E[log(1-D(G(z)))] \quad (1)$$

A função custo (1) tem por objetivo maximizar a probabilidade de classificar corretamente os dados reais e sintéticos enquanto se treina G(.;θ_g) concorrentemente para que se minimize a probabilidade de classificar um dado sintético como tal. Deste modo, é claro de ver que um aumento na capacidade do gerador de produzir amostras sintéticas diminui a capacidade do discriminador de distinguir amostras sintéticas das reais e vice-versa. Isto também mostra a característica adversarial do treinamento. O treinamento para quando o discriminador não consegue mais distinguir imagens sintéticas e reais. Matematicamente, isto ocorrerá quando p_G for igual à p_d, ou seja, o gerador conseguiu fitar os dados perfeitamente.

Uma topologia específica de GAN, a Wasserstein GAN [12], foi utilizada neste trabalho. A Wasserstein GAN foi escolhida pois utiliza como função custo a distância de Wasserstein que é definida como:

$$W(p_d, p_G) = \inf_{\gamma \in \Pi(p_d, p_G)} E_{(x,y)\sim\gamma}[||x-y||] \quad (2)$$

Esta função custo mede a distância entre duas pdfs, mais especificamente, esta função custo mede a quantidade de trabalho necessário que precisa ser realizado para transforma a pdf p_d na pdf p_G. Uma pergunta recorrente é o porquê de não se usar a divergência KL ou a divergência JSD, em [12] se mostrou que a distância KL é mais apropriada para problemas de otimização. Um outro motivo para a escolha desta topologia foi o fato desta topologia ser mais resiliente (pelo menos na prática) ao problema de colapso de modo[2].

IV. Método

Nesta seção descreveremos como as amostras sintéticas foram geradas utilizando as redes GAN e quais foram os métodos utilizados para qualificar tais imagens. De uma maneira geral, neste artigo fazemos análises diretas e indiretas, qualitativas e quantitativas, sobre a verosimilhança das imagens geradas. Neste trabalho, algumas formas diferentes de aferir a qualidade das imagens sintéticas foram usadas, algumas delas medem a qualidade dos dados sintéticos no próprio espaço do dado, outras medem a qualidade dos dados, de maneira indireta, em espaços de projeções desses dados como é o caso dos autocodificadores aonde fazemos a comparação no espaço latente encontrados por aqueles.

A. Pré-Processamento

Como dito anteriormente, este artigo opera sobre o lofargrama produzido pela análise LOFAR. Uma vez que tenhamos produzidos os lofargrams, estes passam apenas por uma normalização antes de serem utilizados para o treinamento das redes neurais. A normalização utilizada foi a ℓ_2, isto é, cada espectro (i.e., cada linha do lofargrama) é dividida pela a norma ℓ_2 daquele espectro.

B. Geração de dados sintéticos

As GAN foram treinadas de duas maneiras diferentes. Na primeira delas separamos as imagens do lofargrama em folds e fazemos validação cruzada com 5 folds. Isto é treinamos uma GAN por fold. Isto foi feito para checar se em alguma partição dos dados o problema de colapso de modo se revelaria. Na segunda maneira, tentamos evitar o problema de colapso de modo treinando uma GAN com todos os dados disponíveis. Nos dois casos GANS especialistas são treinadas, ou seja, treinamos uma GAN especialista para cada classe de navio em estudo, temos uma GAN que foi treinada para gerar somente dados de uma classe, outra para outra classe e assim por diante. Depois do treinamento das GAN ter sido concluído, utilizamos estas para gerar imagens sintéticas de cada classe e estas são utilizadas para avaliar se as GAN conseguiram gerar dados sintéticos com a mesma distribuição dos dados de treinamento (quando mais a frente falarmos que usamos dados sintéticos de um fold significa que utilizamos a GAN treinada naquele fold para gerar dados sintéticos).

Duas perguntas importantes devem ser respondidas concernentes ao treinamento da GAN: a GAN realmente aprendeu a pdf dos dados? A GAN está reproduzindo os dados que viu durante o treinamento ou realmente está criando dados novos ? A primeira pergunta concerne à convergência do treinamento, enquanto que a segunda trata de overfitting.

Estas duas perguntas são de fundamental importância na caracterização dos dados sintéticos. A própria função custo do treinamento das GAN, a distância de Wasserstein, pode responder a primeira pergunta já que como dito a função custo quantifica a distância entre as duas pdfs em questão (as dos dados e a que está sendo estimada pelo gerador) e assim sendo quanto menor a distância de Wasserstein mais parecidas as duas pdf são. Embora a análise da função custo possa responder a primeira pergunta, neste trabalho também foi utilizada a divergência de Kullback-Leiber (KL) para medir se os dados sintéticos seguem a mesma pdf dos dados originais. A ideia é simples: já que cada classe possui diversas tomadas de dados[3], podemos medir a variância estatística de cada classe medindo a divergência KL entre todas estas tomadas para todos os bins de frequência resultantes da análise LOFAR e

[2]O colapso de modo acontece quando a GAN não aprende todos os modos que uma distribuição venha a ter.

[3]Ver seção dados experimentais.

averiguar se a divergência entre os dados reais e sintéticos fica dentro da variância da KL medida nos dados reais.

A análise de componentes principais (PCA) [27] e a sua versão não-linear a Kernel PCA (KPCA) [28] também foram utilizadas para responder essa pergunta, de maneira indireta, através da comparação dos autovalores e de seus autovetores associados, extraídos por essas análises. A ideia é simples, fazemos a PCA nos dados reais e nos dados sintéticos, separadamente, e então, comparamos os autovalores e os autovetores associados extraídos, se os mesmos autovalores (ou próximos) estiverem sendo extraídos isto indicaria que os dados sintéticos são parecidos com os dados reais. Além de extrair a PCA (e a KPCA) com o conjunto de dados completo (com todas as classes de navios), também extraímos as componentes individualmente para cada classe.

A segunda pergunta foi respondida através do uso do erro l2 (definido como a soma dos erros ao quadrado entre os pixels correspondentes de duas imagens). Para cada imagem real no dataset, o erro l2 entre esta imagem e todas as imagens sintéticas (foi utilizado um conjunto de imagens sintéticas com o mesmo número de amostras dos dados reais) foi calculado; se alguma imagem gerada fosse idêntica ou parecida com as imagens sintéticas teríamos o erro l2 em zero ou na sua vizinhança.

C. Principal Component Analysis

A PCA utiliza uma transformação linear e ortogonal para converter um conjunto de observações com variáveis correlacionadas em um espaço com variáveis descorrelacionadas. Estas variáveis descorrelacionadas são chamadas de componentes principais. As componentes principais V são os autovetores da matriz de correlação C dos dados. Estas componentes são ordenadas pela energia dos autovalores, representados na matriz diagonal D, associados a V, sendo que a primeira componente tem a maior energia associada, a segunda a segunda maior energia e assim por diante.

$$V^{-1}CV = D \tag{3}$$

D. Kernel PCA

A maior diferença entre a PCA e a sua extensão não-linear a KPCA é que nesta os autovetores principais são calculados a partir da matriz de kernel (K) ao invés da matriz dos dados C. A motivação do algoritmo KPCA se dá pelo fato de que pontos que não são linearmente separáveis no R^n podem sê-lo no R^d com $d > n$. Para isso, um kernel trick é aplicado que consiste em:

$$K(x, x') = \langle \phi(x), \phi(x') \rangle \tag{4}$$

Aonde $\phi : R^n \rightarrow R^d$. O mesmo procedimento aplicado na PCA é utilizado para a KPCA. Neste trabalho três tipos diferentes de kernel foram testados: rbf, cosseno e sigmoid.

E. KL

Neste trabalho também foi utilizada a divergência de Kullback-Leiber (KL) para medir se os dados sintéticos seguem a mesma pdf dos dados originais já que a divergência KL mensura quão perto estão duas distribuições.

F. Autocodificadores

Os autoencoders [5] ou autocodificadores, são um tipo de rede neural treinada com o objetivo de reproduzir na sua saída os mesmos dados de entrada. A arquitetura deste modelo pode ser interpretada como uma combinação de um codificador que converte os dados de entrada em uma representação mais concisa, e um decodificador que converte a nova representação de volta à representação original. Devido ao fato do autoencoder não precisar de dados etiquetados (já que a saída desejada é igual à entrada) o treinamento deste tipo de arquitetura é visto como não supervisionado. Autoencoders com neurônios totalmente conectados ignoram a estrutura 2D das imagens. Isto é um problema quando se lida com dimensões reais de imagens devido ao grande número de parâmetros, além disso, introduz redundância nos parâmetros forçando que cada neurônio seja conectado de forma global. As camadas convolucionais são uma alternativa as camadas totalmente conectadas para lidar com este tipo de problemas devido ao fato de seus parâmetros serem compartilhados entre cada região da entrada o que reduz o número de parâmetros. O objetivo destas camadas é o de descobrir características locais que se repetem ao longo da entrada.

Dependendo do número de neurônios na última camada do codificador este pode ser classificado como subcompleto ou supercompleto. No primeiro tipo o número de neurônios na camada intermediara é menor em comparação com a dimensão da entrada, caso contrário o autoencoder é considerado como supercompleto. Neste trabalho um autoencoder subcompleto foi treinado utilizando-se apenas dados sintéticos (um autoencoder foi treinado por fold para o caso da GAN treinada utilizando-se folds). A ideia é bastante simples, se os dados reais ativarem, no espaço latente, as mesmas regiões ativadas pelos dados sintéticos, ou seja, se a codificação de uma amostra real e uma amostra sintética for parecida isto indica que os dados reais e sintéticos são parecidos. Além da análise visual, qualitativa, neste trabalho, a divergência KL entre os códigos obtidos pelas amostras reais e sintéticas também é calculada (se a divergência for baixa significa que os dados são parecidos).

V. Resultados Experimentais

Primeiramente, começaremos por descrever os dados experimentais utilizados neste trabalho. Como dito anteriormente este trabalho atua sobre ruídos irradiados por navios. O conjunto de dados utilizados neste trabalho consiste de gravações feitas na raia acústica da Marinha do Brasil em Arraial do Cabo de quatro diferentes classes de navio (identificadas como classe A, B, C e D). Cada gravação foi realizada utilizando-se um único hidrofone omnidirecional posto 45 m abaixo da linha do mar no fundo da raia acústica. Em cada gravação,

chamadas de agora em diante de corridas, um único navio foi posto na raia acústica operando em determinada condição de máquina ao longo de toda a corrida e o ruído acústico emitido por este foi gravado. Este processo foi repetido 10 vezes (menos para a classe A que possui apenas 5 corridas) para todas as classes com diferentes condições de máquina e de mar em cada corrida.

Para cada corrida, o lofargrama desta é calculado. A análise LOFAR utilizou como parâmetros: taxa de decimação igual à 3 e janelas de 1024 amostras no cálculo da STFT. No final do processo cada espectro têm 400 bins de frequência. Deste modo cada lofargrama, será ao final de tudo uma imagem $N \times 400$ aonde N depende da duração de cada corrida. Para cada corrida, o lofargrama correspondente é gerado e então, a partir deste, diversos sub-lofargramas são amostrados a partir dele. Assim como o lofargrama, esses sub-lofargramas também são imagens, porém, de tamanho $L \times 400$. O processo de geração destas sub-imagens começa na primeira linha do lofargrama, L linhas sequenciais são selecionadas para compor uma imagem. Em seguida, a cada R linhas, outras imagens de tamanho $L \times 400$ são geradas da mesma forma. Os parâmetros L (altura da sub-imagem) e R (o tamanho do passo a ser dado entre uma imagem e outra) foram selecionados fazendo-se um grid search aonde se variou L de 5 até 40 de 5 em 5 e R de 1 até 10 de 1 em 1. Os valores que obtiveram os melhores resultados foram 20 e 5 para L e R, respectivamente. Para todas as corridas e todas as classes de navio, o mesmo processo de criação de imagens é repetido. É com este dataset de sub-lofargramas que a GAN é treinada.

O algoritmo de treinamento da Wasserstein GAN utilizado neste trabalho é o WGAN-GP [13]. Tanto a arquitetura do gerador quanto a arquitetura do discriminador da GAN são inspirados na arquitetura da DC-GAN [18]. A GAN foi treinada por 5000 épocas com bateladas de 32 amostras utilizando o algoritmo ADAM [5]. O gerador recebe como entrada um vetor aleatório $v \in R^{100}$ e produz uma imagem 20×400. A arquitetura do gerador consiste em uma camada totalmente conectada, três camadas convolucionais, todas com filtros de tamanho 5×5. O número de filtros usados em cada uma das camadas convolucionais é {64, 32, 1}, respectivamente. Batch-Normalization [19] é aplicada a cada camada na rede, exceto na camada de saída. A técnica de batch-normalization estabiliza o processo de aprendizagem e impede que o gerador colapse todas as amostras em um mesmo ponto conforme descrito em [19]. A função ReLU [5] é usada como função de ativação em todas as camadas, exceto na última camada, que usa a função tangente hiperbólica. O discriminador também é uma CNN. Este recebe como entrada uma imagem de tamanho 20×400 e tem como saída um escalar, aonde 1 significa que a imagem é real e zero significa que esta é sintética. A arquitetura é composta por três camadas convolucionais com filtros de tamanho 5×5 e duas camadas totalmente conectadas. O número de filtros usados nas camadas convolucionais é {16, 32, 32}, respectivamente, e o número de neurônios nas camadas totalmente conectadas é {50, 1}. Mais uma vez, a técnica de batch-normalization é aplicada em todas as camadas, exceto na primeira e na última. Todas as camadas usam as funções LeakyReLU [5], exceto a última, que usa uma função linear.

Os autoencoders como dito anteriormente são compostos de uma etapa de codificação e decodificação. A arquitetura da rede codificadora consiste em 6 camadas convolucionais com funções de ativação LeakyReLU e kernels 3×3. O número de filtros presentes nas camadas convolucionais é igual à {8, 8, 16, 16, 32, 32}. Estas camadas convolucionais são seguidas por duas camadas totalmente conectadas com {128, 2} neurônios, respectivamente, também com funções de ativação do tipo LeakyReLU. A codificação estudada neste trabalho é retirada desta última camada totalmente conectada. A rede decodificadora consiste de duas camadas totalmente conectadas com 128 e 500 neurônios, respectivamente, com funções de ativação LeakyReLU, seguidas por uma camada de reshape (esta camada transforma a saída dos 500 neurônios em uma imagem 5×100) e uma camada de UpSampling (que transforma esta imagem 5×100 em uma imagem 10×200 através de interpolação de zeros). Depois destas camadas seguem duas camadas convolucionais, uma camada de UpSampling (levando à imagem ao tamanho final de 20×400) e 4 camadas convolucionais, todas as camadas convolucionais possuem 32 filtros (menos a última que possui apenas um filtro) com kernels 3×3 e funções de ativação do tipo LeakyReLU. O algoritmo de treinamento utilizado foi o ADAM e a função custo minimizada foi o mse.

Com respeito à geração das imagens, a divergência entre os dados reais e sintéticos fica dentro da variância esperada como pode ser visto na Fig. 4 e na Fig. 5 para a maioria dos bins, sendo que em alguns bins, especialmente nos bins mais ao final do espectro, há divergências. Dado que estes bins costumam representar ruído e não sinal, este não é um problema tão grande.

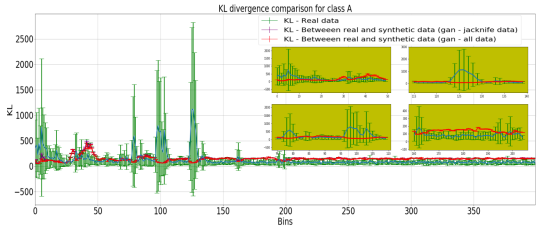

Fig. 4. Comparação entre as divergências de KL dentro da classe A e a divergência de KL entre amostras sintéticas e reais da classe A para as duas GAN treinadas.

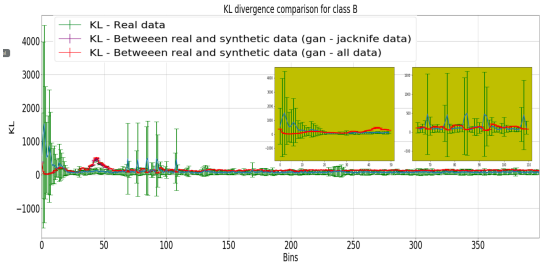

Fig. 5. Comparação entre as divergências de KL dentro da classe B e a divergência de KL entre amostras sintéticas e reais da classe B para as duas GAN treinadas.

Na Fig. 6 temos o erro ℓ_2 entre as imagens reais e sintéticas (tanto as geradas pela GAN treinada com todos os dados quanto as geradas pela GAN treinada utilizando-se folds). O mesmo erro ℓ_2 foi calculado entre as amostras reais para se ter uma base de comparação para o erro esperado. O resultado mostra que o erro ℓ_2 das amostras reais e sintéticas está dentro do esperado e também que nenhuma amostra sintética é uma cópia de uma amostra real.

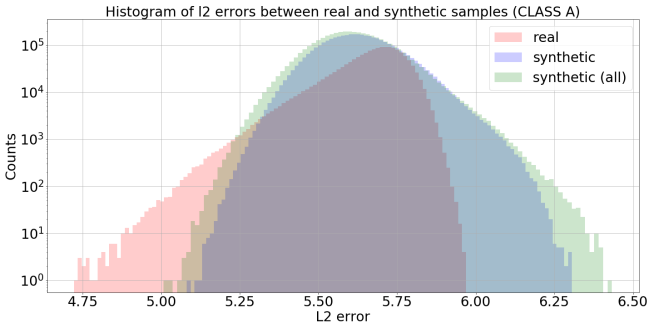

Fig. 6. Erro ℓ_2 calculado *pixelwise* entre as imagens sintéticas e as imagens reais.

Agora apresentaremos os resultdos das análises utilizando-se PCA e KPCA, vale a pena ressaltar que para se aplicar a PCA, o primeiro passo foi vetorizar cada imagem no conjunto de dados de tal modo que a matriz representando a imagem se tornasse um vetor. Na Fig. 7 podemos ver o erro relativo (definimos erro relativo como o valor absoluto entre a razão da diferença entre os autovalores reais (r_{lc}) e sintéticos (s_{lc}) pelos autovalores reais) entre os autovalores obtidos quando se aplica a PCA em dados reais e sintéticos para os dois tipos de GAN, a treinada por folds (jacknife) e a treinada com todos os dados (all) para o caso da extração da PCA com todas as classes.

Na Fig. 8 temos o mesmo gráfico quando a PCA é extraída por classe, no caso em questão, a classe A. A ideia era ver se haveria alguma diferença perceptível quando extraímos uma PCA para cada classe quando comparada com a PCA extraída com todas as classes. Como podemos ver embora haja uma diferença perceptível (principalmente nas primeiras componentes) o erro relativo continua baixo. Porém, maior (e significativo) nas primeiras componentes, todavia o erro relativo é muito baixo nas componentes restantes, o que nos leva a pensar que os dados sintéticos e os dados reais possuem as mesmas componentes principais, o que corrobora o resultado obtido pela análise da KL que indica que os dados sintéticos possuem a mesma pdf dos dados reais.

Na Fig. 9 podemos ver o erro médio quadrático entre as componentes principais extraídas para os dados sintéticos e os dados reais. No caso em questão cada componente tem 8000 entradas (já que cada imagem tem dimensões 20×400) e portanto 8000 componentes são extraídas, assim, calculamos o erro médio quadrático entre cada componente real e sintética, no final obtemos 8000 mse's, um para cada componente. É fácil de observar que o erro entre os autovetores extraídos

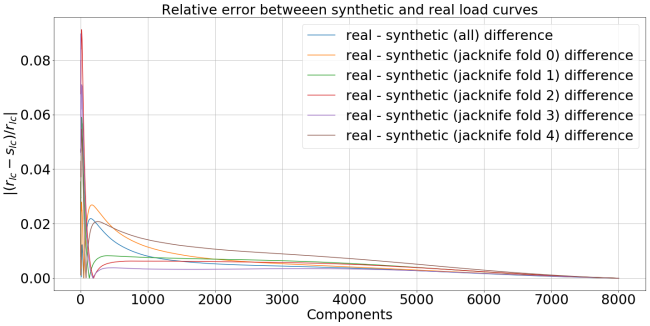

Fig. 7. Erro relativo entre os autovalores quando a PCA é extraída com todas as classes.

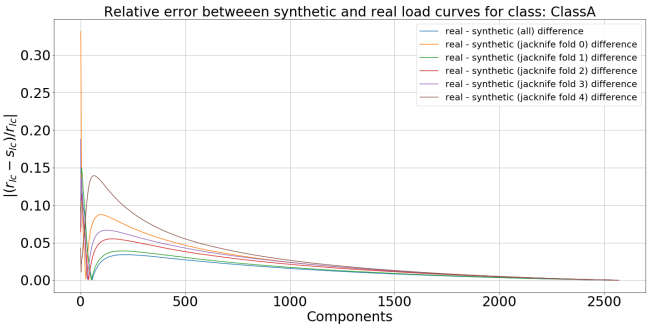

Fig. 8. Erro relativo entre os autovalores quando a PCA é extraída para a classe A.

também é baixo, ou seja, os mesmos autovetores e autovalores associados estão sendo extraídos pela PCA. Para evitar uma repetição desnecessária dos resultados mostraremos o plot apenas para o caso da extração da PCA com todos os dados.

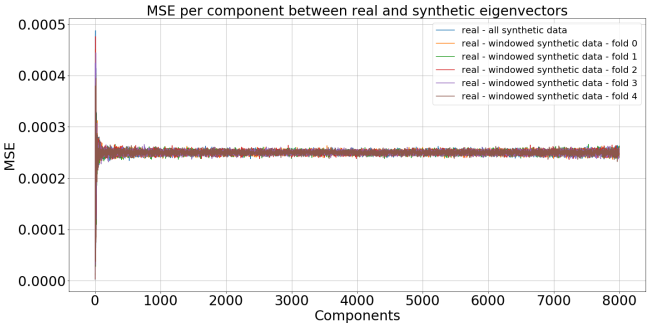

Fig. 9. MSE entre cada componente extraída dos dados reais e dos dados sintéticos.

Os resultados para a KPCA são praticamente idênticos aos resultados obtidos com a PCA clássica independentemente do tipo de kernel utilizado. Devido a este motivo, apresentaremos apenas o resultado para a KPCA com kernel rbf. Os resultados da KPCA também apontam que as mesmas representações dos dados estão sendo extraídas tanto no caso real quanto no caso sintético, o que mais uma vez aponta que a geração de dados sintéticos está seguindo a mesma distribuição de probabilidade dos dados reais.

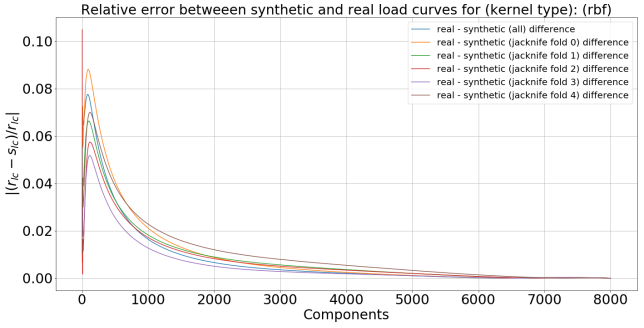

Fig. 10. Erro relativo entre os autovalores quando a KPCA é extraída com todas as classes e com kernel rbf.

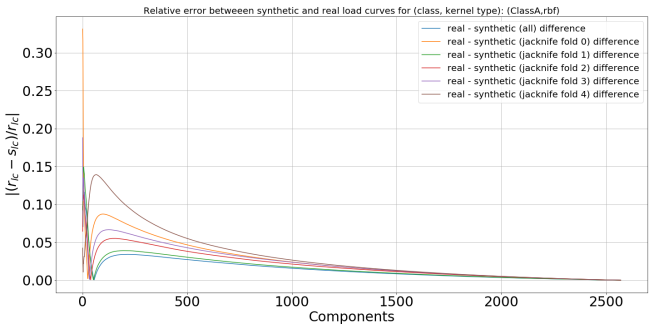

Fig. 11. Erro relativo entre os autovalores quando a KPCA é extraída com a classe A e com kernel rbf.

Nas Figs. 12 e 13 vemos uma tabela com os valores médio e desvio padrão para os erros relativos entre autovalores reais e sintéticos em todos os cenários estudados com a PCA e a KPCA com kernel rbf, respectivamente. Como podemos ver em todos os casos estudados, seja PCA ou KPCA, o erro é baixo.

	ClassA	ClassB	ClassC	ClassD	ALL CLASSES
Fold 1	2.511E-02 +- 2.605E-02	2.086E-02 +- 2.647E-02	3.086E-02 +- 3.661E-02	1.996E-02 +- 2.151E-02	5.613E-03 +- 5.608E-03
Fold 2	1.588E-02 +- 1.589E-02	1.505E-02 +- 1.655E-02	3.681E-02 +- 4.342E-02	2.534E-02 +- 2.892E-02	4.946E-03 +- 4.308E-03
Fold 3	1.988E-02 +- 1.754E-02	2.349E-02 +- 2.934E-02	3.291E-02 +- 3.953E-02	1.620E-02 +- 1.530E-02	4.704E-03 +- 6.392E-03
Fold 4	2.261E-02 +- 2.060E-02	1.935E-02 +- 2.479E-02	3.826E-02 +- 4.874E-02	1.378E-02 +- 1.287E-02	3.039E-03 +- 4.925E-03
Fold 5	3.082E-02 +- 3.394E-02	2.242E-02 +- 2.547E-02	3.490E-02 +- 4.116E-02	2.026E-02 +- 1.956E-02	7.580E-03 +- 5.842E-03
All	1.420E-02 +- 1.359E-02	2.111E-02 +- 2.988E-02	2.466E-02 +- 2.916E-02	1.675E-02 +- 1.647E-02	4.454E-03 +- 4.229E-03

Fig. 12. Tabela comparativa com os valores dos erros relativos entre os autovalores para todas as PCA's, as extraídas por classes e a extraída contendo todas as classes.

	ClassA	ClassB	ClassC	ClassD	ALL CLASSES
Fold 1	1.946E-02 +- 1.877E-02	1.895E-02 +- 2.592E-02	4.710E-02 +- 2.122E-02	1.940E-02 +- 1.077E-02	2.623E-02 +- 1.205E-02
Fold 2	4.816E-02 +- 1.721E-02	1.253E-02 +- 1.492E-02	3.400E-02 +- 2.661E-02	3.833E-02 +- 1.570E-02	3.325E-02 +- 1.302E-02
Fold 3	5.329E-02 +- 1.805E-02	4.118E-02 +- 2.980E-02	2.775E-02 +- 2.723E-02	1.894E-02 +- 1.296E-02	3.529E-02 +- 1.307E-02
Fold 4	3.308E-02 +- 1.906E-02	4.854E-02 +- 1.658E-02	3.418E-02 +- 4.756E-02	3.890E-02 +- 1.354E-02	3.868E-02 +- 6.101E-03
Fold 5	2.549E-02 +- 2.077E-02	5.219E-02 +- 2.000E-02	3.559E-02 +- 2.364E-02	2.335E-02 +- 1.156E-02	3.415E-02 +- 1.139E-02
All	2.373E-02 +- 1.523E-02	1.934E-02 +- 2.272E-02	3.256E-02 +- 1.483E-02	1.324E-02 +- 1.392E-02	2.222E-02 +- 7.038E-03

Fig. 13. Tabela comparativa com os valores dos erros relativos entre os autovalores para todas as KPCA's com kernels rbf, as extraídas por classes e a extraída contendo todas as classes.

Na Fig. 14 temos a projeção, ou código, no espaço latente encontrado por um dos autoencoders treinados. A representação em si não é perfeita já que existe uma zona de confusão entre as classes. Outro aspecto importante que pode ser notado é que a projeção dos dados é altamente não linear, apenas uma classe de navio fica densamente clusterizada em uma região. A ideia original deste artigo era fazer uma comparação dos códigos encontrados utilizando-se a distância do cosseno ou até mesmo a distância euclidiana. Porém, dada as estruturas encontradas no espaço latente essas métricas não fariam muito sentido já que temos amostras de uma mesma classe longe uma das outras. Porém, em regiões aonde não existem nenhuma amostra de outra classe (o que é um efeito desejado deste tipo de projeção). Devido a isso, nos decidimos calcular a divergência KL entre as distribuições de probabilidades definidas pelos códigos de cada classe.

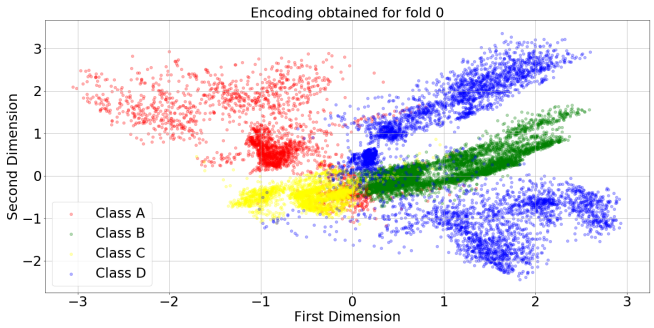

Fig. 14. Projeção dos dados reais no espaço latente encontrado pelo autoencoder.

Como dito anteriormente o autoencoder foi treinado somente com dados sintéticos (para cada fold utilizamos a GAN treinada naquele fold para gerar 48k imagens sintéticas de cada classe e depois treinar o autoencoder) e como um primeiro teste projetamos os dados sintéticos e reais no autoencoder e checamos se os dados ativam as mesmas regiões no espaço latente (esta averiguação visual é possível uma vez que a dimensão de codificação é igual a dois). Podemos ver essa projeção para duas classes na Fig. 15.

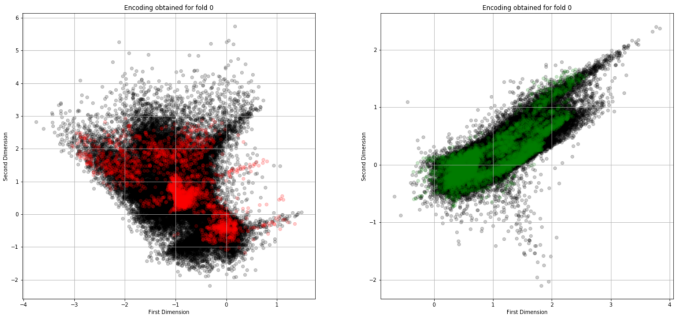

Fig. 15. Projeção dos dados reais e sintéticos no espaço latente encontrado pelo autoencoder para as classes A e B. Os dados em preto são sintéticos.

É importante ressaltar dois pontos importantes nesta figura, o primeiro é que os dados sintéticos ocupam as mesmas regiões que os dados reais e o segundo é que estas regiões ocupadas pelos dados sintéticos são mais densas do que as ocupadas por dados reais. Este último ponto é positivo porque demonstra que os dados gerados pelas GAN não são apenas uma simples cópia dos dados de treinamento e que existe

TABLE I
KL ENTRE AS PROJEÇÕES DOS DADOS REAIS. O X SIGNIFICA NÃO SE APLICA.

	Class A	Class B	Class C	Class D
ClassA	x	5.01 +- 0.45	3.56 +- 0.31	4.57 +- 0.29
ClassB	3.58 +- 0.65	x	3.64 +- 0.90	4.21 +- 0.39
ClassC	2.91 +- 0.36	3.95 +- 1.29	x	4.47 +- 0.50
ClassD	4.29 +- 0.54	4.66 +- 0.27	4.42 +- 0.38	x

TABLE II
KL ENTRE AS PROJEÇÕES DOS DADOS REAIS E DOS DADOS SINTÉTICOS.

	Class A Synt	Class B Synt	Class C Synt	Class D Synt
ClassA	0.37 +- 0.10	7.54 +- 0.64	6.14 +- 0.48	6.90 +- 0.57
ClassB	6.02 +- 0.89	0.15 +- 0.04	5.29 +- 1.22	4.84 +- 0.46
ClassC	3.29 +- 0.40	5.09 +- 1.97	0.37 +- 0.10	5.16 +- 1.00
ClassD	7.14 +- 1.02	6.29 +- 0.60	7.24 +- 0.65	0.28 +- 0.08

sim uma variabilidade nos dados gerados. Para checar se os dados reais e os dados sintéticos possuem a mesma pdf, mais uma vez, recorremos à divergência KL. Nas Tabelas I e II vemos a divergência KL entre os dados reais em si (esse teste serve como uma medida do limite de similaridade que a GAN pode alcançar neste espaço de projeção) e entre os dados sintéticos e reais. A ideia é checar se a divergência entre dados sintéticos e reais é parecida com a divergência entre dados reais e também checar se a divergência intraclasse é baixa. Como dito anteriormente, para cada fold uma GAN foi treinada, utilizando-se essa GAN geramos dados sintéticos e treinamos um autoencoder, depois medimos a KL entre as projeções de todos os folds, i.e., para cada fold projetamos os dados, tanto reais quanto sintéticos, no autoencoder treinado e medimos a KL entre estas projeções, gerando assim uma média e uma barra de erro.

Como podemos ver a divergência KL entre dados sintéticos e reais de uma mesma classe é baixa e entre dados de classes distintas alta. Percebemos também que os valores das divergências KL entre dados reais e dados reais e sintéticos são bem próximos. Este resultado corrobora ainda mais o resultado obtido pela KL entre as imagens sintéticas e reais, sendo que aqui, mesmo depois da projeção em um espaço latente a KL continua baixa.

VI. CONCLUSÃO

Neste artigo propusemos o uso de redes neurais adversariais para a sintetização de dados de um sistema de sonar passivo. Como podemos ver os dados sintéticos se mostraram verossímeis dentro das análises feitas. Como trabalho futuro pretendemos continuar explorando a verossimilhança desses dados sintéticos e a utilização dos mesmos no treinamento de modelos classificadores profundos. Podemos também aprofundar a análise conduzida no autoencoder para diferentes dimensões de codificação e tentar utilizar algoritmos de clusterização sobre os dados projetados no espaço latente como uma forma de classificação.

AGRADECIMENTOS

Os autores gostariam de agradecer ao CNPq e à FAPERJ pelo apoio a este trabalho. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior Brasil (CAPES).

REFERENCES

[1] R. Urick, Principles of Underwater Sound for Engineers. McGraw-Hill.
[2] R. P. Hodges, Underwater acoustics: Analysis, design and performance of sonar. John Wiley & Sons, 2011.
[3] R. O. Nielsen, Sonar Signal Processing. Norwood, MA, USA: Artech House, Inc
[4] W. S. Burdic, Underwater Acoustic System Analysis. Peninsula Pub.
[5] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press.
[6] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Networks, vol. 61, no. Supplement C, pp. 85 - 117.
[7] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, Nature, vol. 521, no. 7553, pp. 436 - 444, 2015.
[8] Q. Li, Digital Sonar Design in Underwater Acoustics. Springer-Verlag Berlin Heidelberg, 2012.
[9] W. Soares-Filho, J. M. Seixas, and L. P. Caloba, Enlarging neural class detection capacity in passive sonar systems, in 2002 IEEE International Symposium on Circuits and Systems. Proceedings, 2002.
[10] M. Maschler, E. Solan, & S. Zamir, Game Theory. Cambridge: Cambridge University Press, 2013.
[11] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, Generative Adversarial Networks, 2014.
[12] M. Arjovsky, S. Chintala, L. Bottou 2017. Wasserstein GAN.
[13] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A. Courville 2017. Improved Training of Wasserstein GANs.
[14] N. N. de Moura Junior, Detecção de Novidades para sistemas de sonar passivo. Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, Brasil, 2018.
[15] W. Soares-Filho, J. M. Seixas, and L. P. Caloba, Enlarging neural class detection capacity in passive sonar systems, in 2002 IEEE International Symposium on Circuits and Systems. Proceedings 2002.
[16] Q. Li, Digital sonar design in underwater acoustics: principles and applications. Springer Science & Business Media, 2012.
[17] G. Gong, S. U. D. of Statistics, Cross-validation, the Jackknife, and the Bootstrap: Excess Error Estimation in Forward Logistic Regression, Stanford University, 1982.
[18] A. Radford, L. Metz, S. Chintala 2015. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.
[19] S. IOFFE, C. SZEGEDY, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015.
[20] R. O. Nielsen, Cramer-Rao lower bounds for sonar broad-band modulation parameters, 1999, IEEE Journal of Oceanic Engineering.
[21] FIORE, U., DE SANTIS, A., PERLA, F., et al. "Using Generative Adversarial Networks for Improving Classification Effectiveness in Credit Card Fraud Detection", Information Sciences, 12 2017.
[22] ZHENG, Y., ZHOU, X.-H., SHENG, W.-G., et al. "Generative adversarial network based telecom fraud detection at the receiving bank", Neural Networks, v. 102, 03 2018.
[23] SCHLEGL, T., SEEBÖCK, P., WALDSTEIN, S. M., et al. "Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery", CoRR, v. abs/1703.05921, 2017
[24] LIU, M., YUAN, F., ZHU, Y., et al. "Generating Underwater Images by GANs and Similarity Measurement". In: 2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO), pp. 1–5, May 2018
[25] XU, Y., ZHANG, Y., WANG, H., et al. "Underwater image classification using deep convolutional neural networks and data augmentation". In: 2017 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC), pp. 1–5, Oct 2017
[26] SUNG, M., KIM, J., YU, S. "Image-based Super Resolution of Underwater Sonar Images using Generative Adversarial Network". In: TENCON 2018 - 2018 IEEE Region 10 Conference, pp. 0457–0461, Oct 2018.
[27] A. Hyvärinen and E. Oja. 2000. Independent component analysis: algorithms and applications. Neural Netw. 13, 4-5 (May 2000), 411-430.
[28] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. 1998. Nonlinear component analysis as a kernel eigenvalue problem. Neural Comput. 10, 5 (July 1998), 1299-1319.