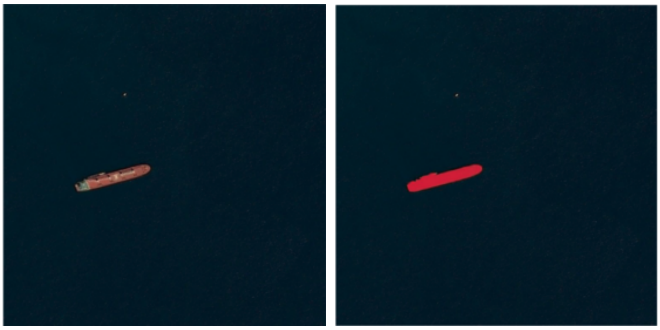
Projeto Final

Rodrigo Rusa
208592
rodrigorusa@gmail.com

Rogério Rocha
191084
rogerio.rocha@ic.unicamp.br

Thales Nazatto
074388
tenazatto@gmail.com

***Resumo*—O transporte marítimo é um dos meios mais utilizados no comércio internacional. Devido a este fato, autoridades governamentais e agências de proteção ambiental vigiam constantemente as encostas onde se concentram muitas dessas embarcações. A *Airbus* tem oferecido serviços de monitoramento que auxiliam o setor marítimo, cujo objetivo é extrair objetos de imagens de satélite automaticamente. Neste projeto, propomos explorar técnicas de aprendizado de máquina e reconhecimento de padrões para classificar imagens, além de detectar e segmentar a localização de navios nessas imagens. Apresentamos os resultados obtidos para cada arquitetura proposta e avaliamos os modelos com diversas métricas encontradas na literatura. Por fim, apresentamos nossas conclusões e sugerimos ainda explorar outras técnicas que suspeitamos melhorar os resultados obtidos.**



(a) Imagem de satélite original. (b) Detecção do navio.

Figura 1. Comparação da imagem de entrada com a imagem de saída.

I. Introdução

Um dos meios de transporte mais utilizados no comércio internacional, o transporte marítimo, pode englobar todo tipo de cargas desde químicos, combustíveis, alimentos, areias, cereais, minérios e automóveis. O uso de contêineres como meio de empacotamento de carga contribuiu para o desenvolvimento do transporte marítimo desde a década de 1960. O aumento do número de navios aumentam as chances de infrações no mar, como por exemplo, acidentes com embarcações, pirataria, pesca ilegal, tráfico de drogas e movimentação ilegal de cargas, algumas delas ambientalmente devastadoras. Isso obrigou autoridades governamentais e agências de proteção ambiental nacionais a vigiarem suas encostas. A *Airbus* oferece serviços abrangentes de monitoramento marítimo com detalhes precisos e monitoramento intensivo, combinando dados de satélites com o trabalho de analistas altamente treinados que auxiliam o setor marítimo antecipando ameaças, acionando alertas e aumentando a eficiência no mar. Muito trabalho foi feito nos últimos 10 anos para extrair objetos automaticamente a partir de imagens de satélite com resultados significativos, porém, sem efeitos operacionais efetivos.

II. Objetivos e Ferramentas

Este trabalho tem como objetivo encontrar o melhor modelo com base em técnicas de aprendizado de máquina e reconhecimento de padrões que possa detectar e extrair a localização de navios a partir de imagens de satélite, conforme ilustrado na Figura 1.

A base de dados a ser utilizada neste projeto encontra-se no Kaggle [1]. Essa base de dados possuía a princípio 104.070 imagens para treinamento e 88.500 para teste, no entanto, no meio do projeto as frações de treinamento e teste foram alteradas para 192.566 e 15.163 imagens, respectivamente. Todas as imagens possuem tamanho de 768×768 pixels, colorida e em formato *.jpg*, as quais podem conter ou não navios, indicados em um arquivo extra em formato *.csv* que contém o identificador da imagem (nome do arquivo) e um identificador de presença ou não de navios. O rótulo de saída esperado é uma máscara com os navios detectados em cada imagem codificada com *Run-Length*. Muitas imagens não contêm navios e as que contêm podem conter vários navios. Navios dentro e através de imagens podem diferir em tamanho (às vezes significativamente) e estar localizados em mar aberto, em docas, marinas, etc. Para essa métrica, os segmentos de objeto não podem se sobrepor. Havia uma pequena porcentagem de imagens no conjunto treino e teste que tinham uma pequena sobreposição de segmentos de objeto quando os navios estavam diretamente próximos um do outro. Quaisquer sobreposições de segmentos foram removidas.

Para a realização deste trabalho, foram utilizadas a linguagem de programação *Python 3* e as bibliotecas essenciais para manipulação de arquivos (**pandas**), matemática (**numpy**) e (**scipy**), gráfica (**matplotlib**), pré e pós processamento (**sklearn**) e (**keras.preprocessing.image**), de imagens (**skimage**), de redes neurais (**sklearn.neural_network**), e de redes neurais profundas convolucionais (**keras.models**), (**keras.layers**), (**keras.layers.core**) e (**keras.applications**).

III. Metodologia e Métodos

Apresentamos a seguir, a metodologia e métodos que utilizamos para alcançar o principal objetivo desse projeto. Começamos com técnicas de redes neurais para *baseline*, depois

avançamos à técnicas mais complexas com redes neurais profundas.

A. Baseline - Redes Neurais

Primeiramente, antes de analisarmos os dados, simplificamos o objetivo principal do problema que consistia em detectar e extrair a posição dos navios nas imagens de satélite para um novo objetivo na *baseline* que consistia apenas em determinar se uma imagem continha ou não navios, i. e., problema de classificação. Para isso, adicionamos uma *feature* com valores inteiros entre 0 e 1, onde 0 representa a ausência de navios e 1 a presença de navios. Na próxima etapa, a fim de construir nossa solução, analisamos os tipos de dados encontrados na base utilizada. Como os dados estavam em formato *.jpg* vetorizamos cada imagem e construímos um arquivo *.csv* contendo os dados de treinamento com 10407 imagens, 10% do total da base de dados e um único arquivo *.csv* com 50% do total de imagens para testar nosso melhor modelo, além disso, reduzimos o tamanho da imagem de 768×768 para 64×64 e para 128×128 devido a quantidade de memória disponível nas máquinas para as experimentações, que não era suficiente para acomodar as *features* da imagem original. A partir deste ponto, geramos arquivos *.csv* com imagens de dimensões 64×64 e 128×128 para imagens em RGB e escala de cinza - onde fizemos um *downscale* pela média do filtro, ainda, transformamos a imagem em escala de cinza através da transformada discreta do cosseno (DCT - *Discrete Cossine Transform*) e também através da transformada de *Fourier* - em que utilizamos a implementação da transformada rápida de *Fourier*, (FFT - Fast Fourier Transform). Em todos os treinamentos, normalizamos os dados da base dividindo pelo valor máximo (255) admitido em cada *feature*, exceto quando aplicado as transformadas DCT e FFT.

Para encontramos o melhor modelo que detectasse a presença de navios nas imagens, utilizamos a técnica de aprendizado de máquina e reconhecimento de padrões conhecida como redes neurais. Utilizamos a implementação *MLPClassifier*, encontrada na biblioteca do *Scikit-learn*, este modelo otimiza a função de perda *log* usando a descida de gradiente estocástico [2]. Nossas arquiteturas propostas foram com duas camadas escondidas, a primeira arquitetura foi configurada com 2048 unidades de neurônios na camada 1 e com 1024 unidades na camada 2 para imagens com dimensões 64×64, em nossa segunda arquitetura, para imagens com dimensões 128×128, utilizamos 4096 neurônios na camada 1 e 2048 unidades na camada 2. Utilizamos como função de ativação a *ReLU*, um número máximo de iterações de 1000 iterações, a taxa de aprendizado padrão com valor de 10^{-3}, e todos os outros parâmetros como *default* da biblioteca.

Os modelos foram validados utilizando a técnica *k-fold Cross Validation*, que divide os dados em *k* partes, sendo uma dessas partes utilizadas para validação e o restante para treinamento. O processo é repetido *k* vezes, em cada iteração alternamos os dados entre validação e treinamento [2]. Antes da divisão, foi realizado um embaralhamento dos dados, para que se evitasse a má distribuição dos dados em cada partes, o que acarretaria em resultados ruins. Assumimos o valor 5 para *k* utilizado para o particionamento dos dados e iterações. A cada iteração, armazenamos os melhores parâmetros de acordo com o menor erro obtido no treinamento e melhor acurácia obtida na predição dos dados de validação.

As métricas utilizadas para avaliar os modelos foram a acurácia normalizada, a precisão, o *recall* e o *F1 score*. Para a *baseline* só utilizamos as imagens de treinamento para validação dos modelos, a base de teste não foi utilizada para ser usada apenas ao final do projeto no melhor modelo encontrado.

B. Redes Neurais Profundas - CNN's

No intuito de melhorar os resultados obtidos pelas redes neurais utilizadas na *baseline*, utilizamos técnicas de *deep learning*, como redes neurais convolucionais (*Convolutional Neural Network* - CNN) [2]. As imagens foram mantidas em formato *.jpg* como entrada para as CNN's, mas agora utilizando o novo conjunto de treinamento disponibilizado com 192.566 imagens.

A primeira rede ou arquitetura proposta contém 3 camadas de convolução, duas com 32 mapas de ativação e filtros de 3×3 e uma com 64 mapas de ativação e filtros de 3×3, e a função *ReLU* para ativação, seguidas por camadas de *Max-Pooling* com tamanho 2×2, e uma camada completamente conectada ou densa, modelo muito similar à *LeNet* proposta por Yann LeCun, Léon Bottou e Yoshua Bengio em 1998. A Figura 2 traz uma ilustração dessa rede.

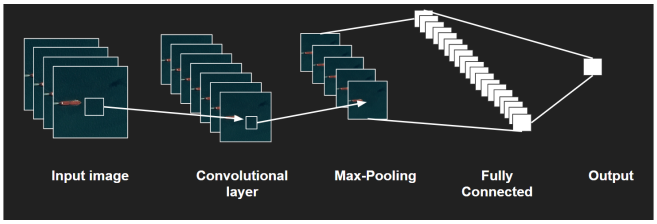


Figura 2. Arquitetura de uma CNN similar à LeNet.

Utilizamos imagens com dimensões 128×128 e 256×256 como entrada para essa rede, e treinamos modelos para 10 e 50 mil imagens de entrada, todas em RGB, aproximadamente 5% e 25% do conjunto de treinamento, respectivamente. Para compilar nossa arquitetura, utilizamos como parâmetros o otimizador *msprop*, otimizando a função de perda *Binary Cross Entropy* e a métrica de acurácia. Treinamos nosso modelo com o tamanho do *batch* de 100 imagens e utilizamos 50 épocas para a convergência.

A segunda rede CNN avaliada foi a ***SqueezeNet*** desenvolvida por Forrest Iandola et al.em 2017, uma pequena arquitetura, simplificação da *AlexNet* proposta por Alex Krizhevsky, IIya Sutskever e Geoff Hinton em 2012, que apresenta alta acurácia para a base de dados do *ImageNet*. Esta arquitetura foi experimentada devido ao ótimo resultado apresentado na tarefa 4 da disciplina para a base CIFAR10. A Figura 3 mostra a arquitetura dessa rede.

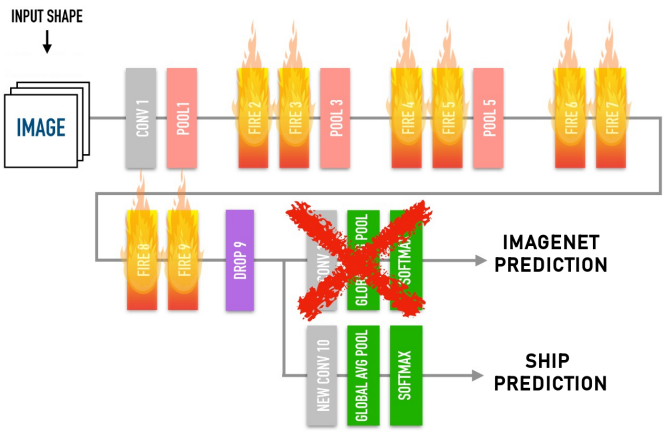


Figura 3. Arquitetura da SqueezeNet.

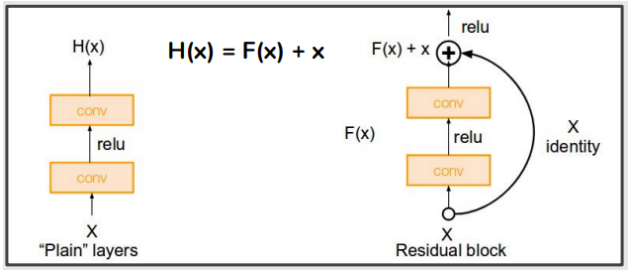


Figura 4. Funcionamento da ResNet.

As imagens utilizadas no treinamento tinham dimensões 128×128, treinando modelo com 10 mil imagens de entrada, também em RGB. A compilação dessa arquitetura e treinamento do modelo utilizou os mesmos parâmetros que a rede anterior, modificando apenas o otimizador para a descida de gradiente estocástico. Realizamos uma transferência de aprendizado nessa rede, utilizando os pesos do modelo treinado sobre a base de dados do *ImageNet*, no intuito de obter melhores resultados e menor tempo de processamento, uma vez que não temos muitos dados disponibilizados para treinarmos uma rede CNN do zero. A maioria dos filtros aprendidos em camadas convolucionais iniciais detectam elementos de baixo nível, como bordas, cantos e *blobs* de cores, que são comuns à maioria dos problemas no domínio da imagem, enquanto que as camadas superiores antes da classificação, detectam elementos específicos dos objetos.

A terceira rede convolucional testada em nossa base de dados foi a ***VGG16***, proposta por Karen Simonyan e Andrew Zisserman em 2014. Esta arquitetura é mais profunda que a *AlexNet*, no sentido de ter mais camadas, 16, para extração de características, possuindo filtros com dimensões 3×3 para a convolução, menores que esta última arquitetura. Realizamos também nessa arquitetura uma transferência de aprendizado em vários níveis de camadas (classificação, específicas e todas), utilizando os pesos do modelo treinado sobre a base de dados do *ImageNet*. Também utilizamos 10 mil imagens para um treinamento inicial, com imagens de dimensões 128×128 e parâmetros de compilação e treinamento iguais à rede anteriormente avaliada.

Por fim, a última rede que utilizamos para encontrarmos nosso modelo foi a ***ResNet-50*** proposta por Kaiming He et al. em 2015. Redes derivadas da *ResNet* tem a característica peculiar de serem bem profundas, chegando até 1000 camadas de profundidade. Para lidar com essa imensa profundidade, usam camadas de rede para ajustar um mapeamento resudual em vez de tentar diretamente ajustar um mapeamento subjacente desejado, como mostra a Figura 4.

A arquitetura completa da *ResNet* consiste de pilhas de blocos residuais, camadas de convolução 3×3 em cada bloco residual e apenas uma camada completamente conectada para a classificação. Também, são usadas camadas com "gargalo"(*bottleneck*) para melhora de eficiência como a arquitetura *GoogLeNet* proposta por Szegedy et al. em 2014. Assim como nas arquiteturas anteriores, utilizamos 10 mil imagens para um treinamento inicial, com imagens de dimensões 128×128, parâmetros de compilação e treinamento não alterados e realizamos transferência de aprendizado em vários níveis de camadas, utilizando os pesos do modelo treinado sobre a base de dados do *ImageNet*.

C. Detecção e Segmentação

Para detectar e extrair a posição de navios nas imagens de satélite, continuamos com técnicas de *deep learning*, porém adotamos um *framework* muito utilizado para esse fim, e que foi utilizado na competição da *Airbus Ship Detection* por muitos dos competidores.

A detecção de navios nas imagens de satélites consiste em um problema de segmentação de instâncias, uma das tarefas mais difíceis de visão computacional que consiste na identificação dos contornos de cada objeto no nível de pixel. Os modelos adotados para resolver tal problema são divididos em dois passos principais, primeiro, a detecção de objetos relevantes demarcando-os por *bounding boxes*, em seguida é realizada a segmentação semântica para cada *bounding box* detectada selecionando quais pixels pertencem ao objeto (*foreground*) e quais pertencem ao fundo (*background*). Este modelo pode ser implementado em diversos *frameworks*, e um desses *frameworks* é conhecido como *Mask R-CNN* [3]–[5].

O *Mask R-CNN*, em um primeiro estágio, realiza uma varredura (*scan*), na forma de janelas deslizantes, na imagem detectando áreas de objetos, e posteriormente, classifica essas áreas gerando *bounding boxes* e suas devidas máscaras. Este *framework* é uma extensão do *Faster R-CNN* utilizada para detecção de objetos, em que foi adicionado às duas saídas já existentes na *Faster R-CNN*, **classe do objeto** e ***box offsets***, uma terceira saída correspondente a máscara, i. e., uma ***Fully Convolutional Network*** para segmentação semântica, pixels dos objetos em cada instância. O *Mask R-CNN*, assim como o *Faster R-CNN*, utiliza as redes ResNet (50 ou 101) em seu *backbone* para extração de *features* das imagens e detecção de objetos.

Os *bounding boxes* encontrados após a varredura da imagem são refinados usando um modelo de regressão linear.

Os principais módulos desse framework são:

- **Backbone**: uma rede neural convolucional (CNN) padrão que serve como extrator de *features*, são elas:
  - *ResNet (50 ou 101)*;
  - *Feature Pyramid Network (FPN)*: A FPN melhora a extração de features piramidal padrão pois adiciona uma segunda pirâmide que contém features de alto nível da primeira pirâmide e passa essa informação para as camadas mais baixas.
- **Region Proposal Network (RPN)**: é uma rede neural que faz o escaneamento da imagem na forma de janelas deslizantes encontrando áreas onde contém objetos. A RPN gera duas saídas para cada região (*anchor*):
  - Classe do *anchor*: *background* ou *foreground*;
  - Refinamento do *bounding box*: valor para refinar o ajuste da caixa ao objeto detectado.
- **ROI Pooling**: reajuste do tamanho dos *bounding boxess* para um tamanho fixo. No *Mask R-CNN* é usado o método chamado *ROI Align*.
- **Segmentation Masks**: Se parássemos na seção anterior teríamos o *framework Faster R-CNN* para detecção de objetos. Dessa forma, para a detecção das máscaras de cada instância proposta no *framework Mask R-CNN* foi adicionada uma rede convolucional que pega regiões positivas selecionadas pelo classificador de *ROIs* e gera suas respectivas máscaras em baixa resolução 28×28 pixels.

A Figura 5 nos mostra a arquitetura do *framework Mask R-CNN* para uma melhor ilustração do modelo.

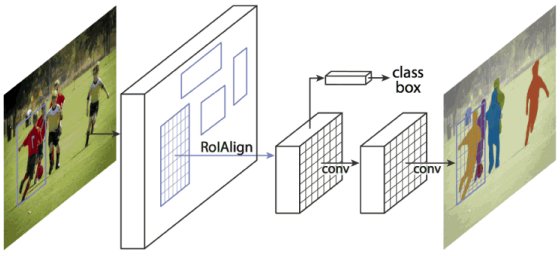


Figura 5. O Mask R-CNN para segmentação de instâncias.

Para calcular a Precisão e Recall dos objetos detectados é utilizada a medida Intersecção sobre a União (IoU) [6], ilustrada na Figura 6, que diz o quão próximo a área detectada é em relação ao *ground truth*.

Desse modo, aplicando um threshold entre $0,5$ e 1 podemos dizer se a detecção acertou ou errou, como ilustra a Figura 7. Neste trabalho, foi utilizado $0,5$, ou seja, se a razão dada acima foi maior que $0,5$ então a detecção foi precisa.

Para avaliar a acurácia dos modelos foi utilizada a medida *mean Average Precision (mAP)* [6] que é bastante empregada em problemas de detecção. Essa medida consiste na média das precisões máximas para diferentes valores de *Recall*.

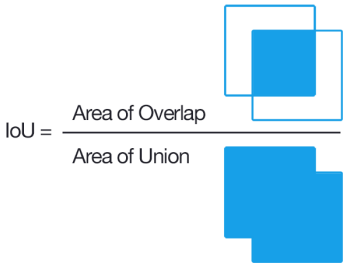


Figura 6. A métrica Intersecção sobre União.

Figura 7. Precisão da Intersecção sobre União.

Em outras palavras, ela computa a área embaixo da curva de *Precisão versus Recall*, pois uma detecção é considerada acurada se a *Precisão* permanece alta com o aumento do *Recall*. Para a geração da curva os dados são interpolados no intervalo de 0 a 1 para a medida de *Recall*.

A princípio, executamos alguns experimentos modificando a rede do *backbone* do *framework Mask R-CNN*, escolhendo o melhor resultado para dar prosseguimento aos experimentos. Posteriormente, experimentamos a técnica de *Transfer Learning* utilizando os pesos da base *MS COCO* treinada no *framework Mask R-CNN*. Tal base, *MS COCO*, é muito utilizada na literatura para resolução de problemas de visão computacional tais como detecção e segmentação de objetos. Testamos a transferência de aprendizado em diferentes níveis desde um *fine tuning*, iniciando com apenas as últimas camadas (heads) até um *fine tuning* de todas as camadas. Para tentar melhorar os resultados obtidos fizemos aumentação dos dados gerando imagens rotacionadas, com *flip* e aplicando alguns filtros tais como Filtro Gaussiano. Como muitas imagens possuíam navios que ocupam poucos *pixels* na imagem, e como utilizamos imagens com tamanho reduzido, 256×256, esses navios acabam sendo perdidos na imagem e não sendo detectados pela rede. Dessa forma, treinamos também a rede utilizando as imagens com a resolução original de 768×768 *pixels*. A princípio, utilizamos para treinamento apenas as imagens que continham navios, cerca de 42 mil imagens, 22% do total de imagens para treinamento. Ao final, obtemos nosso melhor modelo treinando com a base de dados completa para treinamento.

Na próxima seção, apresentaremos os resultados obtidos com a aplicação dessa metodologia.

IV. Experimentos e Discussão

Os experimentos foram organizados em três etapas, de acordo com a metodologia introduzida na seção anterior. Primeiramente, apresentamos os resultados obtidos para o *baseline* que teve foco nas redes neurais e extração de características das imagens de satélite. Em seguida, serão mostrados os experimentos das redes neurais profundas, com foco em arquiteturas de CNN's. E por fim, traremos os resultados obtidos para detecção e segmentação, que teve como foco o *framework Mask R-CNN* e técnicas comuns de aprendizado de máquina para melhoras de resultados.

A. Baseline

Dividimos os experimentos em dez partes, em que apresentamos os resultados para cada solução proposta, imagens de dimensões 64×64 (primeiros cinco experimentos) e 128×128 (cinco últimos experimentos), ambas com 10% do total de imagens de treinamento e em RGB, escala de cinza, DCT, FFT, além disso, 64×64 em DCT RGB com 10407 imagens e 128×128 em DCT com 52035 imagens, 50% da base de dados de treinamento. Em cada parte, apresentamos o modelo com menor erro e maior acurácia obtido da técnica *k-fold Cross Validation*. Para nosso melhor modelo, apresentamos um gráfico da convergência da função de custo com a variação da quantidade de iterações, uma tabela da matriz de confusão para as duas classes do problema, e ainda, algumas imagens que nosso melhor modelo acertou e errou da base de dados para validação.

Na Tabela I podemos constatar que não há diferenças significativas no custo e acurácia entre os tipos RGB e Escala de cinza, tanto com dimensões de 64×64 quanto com 128×128. Quando aplicamos a DFT na imagem em escala de cinza, temos uma melhora de mais ou menos 5% na acurácia com 64×64 e de 12% com 128×128. Ao aplicarmos a DCT na imagem em escala de cinza, aumentamos a acurácia em 14%, tanto com 64×64 quanto com 128×128. Utilizamos, dessa forma, o modelo gerado para as imagens em DCT com dimensões de 128×128 para avaliarmos com uma base maior de dados para treinamento, 50% da base original, acurácia aumentou em 3 pontos percentuais, um resultado muito melhor que o primeiro experimento. Apesar da acurácia ser de aproximadamente 70% com as imagens em RGB e em Escala de cinza, a precisão mostra que os modelos não são bons, acertando pouco mais de 50% da classe 1, imagens que tem navios, em RGB, e 0% dessas imagens quando utilizadas em Escala de cinza. Por outro lado, os modelos das imagens em DCT mostram um equilíbrio na precisão, com alta taxa de acertos, superior a 80%. De forma semelhante, quando utilizamos a métrica *Recall*, vemos um desequilíbrio muito grande no acerto de classes quando utilizados os modelos para RGB, Escala de cinza e DFT. A métrica *F1 score* resume os resultados da precisão e *Recall* apresentadas.

A Figura 8 nos mostra o comportamento do custo da função de perda *log* em relação à quantidade de iterações utilizadas para a convergência do método para encontrar o melhor modelo com 50 mil imagens de treinamento. Como pode ser

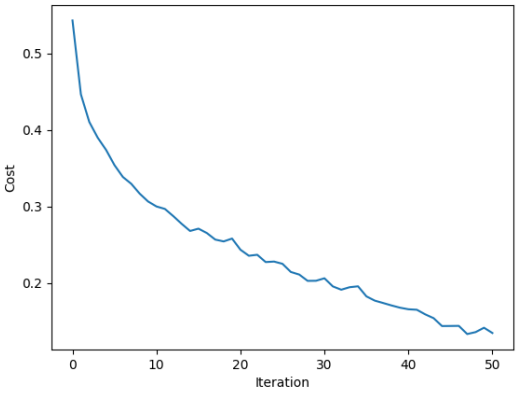


Figura 8. Convergência da função de custo para o melhor modelo.

observado, o custo da função alcança seu mínimo aproximado com cerca de 50 iterações. Na Figura 9, podemos ver a matriz de confusão normalizada também para este melhor modelo encontrado. Pelos valores apresentados, podemos constatar uma alta taxa de acertos para as duas classes.

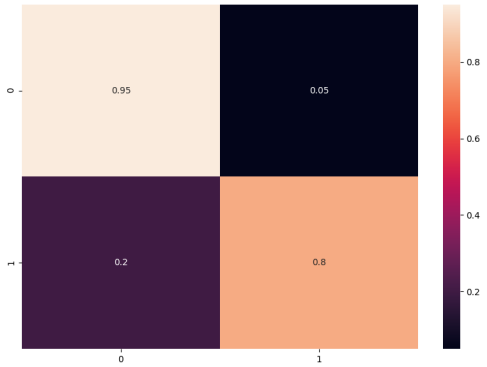


Figura 9. Matriz de confusão para o melhor modelo.

Na Figura 10, podemos ver os tipos de imagens que o modelo acerta, imagens mais limpas em que o navio se destaca, e os que não acertam, imagens mais sujas com nuvens, cor de mar não padrão e com grandes faixa de terra.

B. Redes Neurais Profundas - CNN's

Os experimentos com as CNN's (*Convolutiona Neural Network's*) foram organizados em quatro etapas, conforme à metodologia, onde primeiramente foi avaliada uma CNN implementada muito similar à *LeNet*, em seguida, fizemos um *fine tune* na arquitetura *SqueezeNet*, e por fim, obtemos modelos realizando transferência de aprendizado também com as arquiteturas *VGG16* e *ResNet-50*. Em cada parte, apresentamos os resultados obtidos das métricas, acurácia normalizada, precisão, *recall* e *F1 score*. Para nosso melhor modelo, apresentamos um gráfico da convergência da função

Tabela I
COMPARAÇÃO DOS RESULTADOS DOS EXPERIMENTOS DO *baseline*

Experimento	Tipo	k-fold	Custo	Acurácia	Precisão		Recall		F1 score	
					0	1	0	1	0	1
1	RGB	k2	0,489	0,7116	0,7171	0,575	0,9768	0,075	0,827	0,1329
2	Escala de cinza	k2	0,575	0,7059	0,7059	0	1	0	0,8276	0
3	DCT	k5	0,281	0,8543	0,8662	0,8159	0,9387	0,6519	0,901	0,7247
4	FFT	k2	0,463	0,7693	0,7654	0,8027	0,9707	0,2859	0,8559	0,4216
5	DCT (RGB)	k2	0,417	0,814	0,8091	0,8398	0,9639	0,4542	0,8797	0,5896
6	RGB	k1	0,497	0,7284	0,7658	0,5616	0,8863	0,3496	0,8217	0,431
7	Escala de cinza	k2	0,574	0,7059	0,7059	0	1	0	0,8276	0
8	DCT	k1	0,026	0,8731	0,8966	0,8096	0,9271	0,7434	0,9116	0,7751
9	FFT	k3	0,214	0,8563	0,8679	0,8187	0,9394	0,6568	0,9022	0,7289
10 (Melhor modelo)	DCT	k4	0,134	0,9084	0,925	0,861	0,95	0,8008	0,9373	0,8298

Figura 10. Comparação de imagens com navios detectados e não detectados para *baseline*

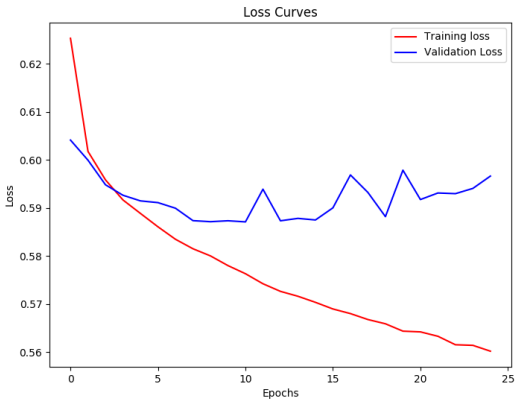


Figura 11. Convergência da função de custo para o melhor modelo com as CNN's.

de custo com a variação da quantidade de épocas, um gráfico mostrando o comportamento da acurácia com a variação das épocas, e ainda, algumas imagens que nosso melhor modelo acertou e errou da base de dados para o teste.

Na Tabela II podemos ver que os resultados obtidos com a CNN implementada giram em torno dos 90% de acurácia, o que é considerado um ótimo resultado, mesmo variando a quantidade de imagens de 10000 para 50000 e o tamanho dessas de 128×128 para 256×256. Os resultados também são similares com as outras métricas avaliadas. Por outro lado, os resultados apresentados pelo uso da arquitetura *SqueezeNet* e *ResNet-50* não foram considerados satisfatórios, com valores abaixo dos 80%, mesmos realizando um *fine tune* em diversas camadas. Isto pode ser melhor constatado, quando observamos os resultados das métricas: precisão, *recall* e *F1 score*, que apresentam valores 0 e 1 indicando que só existem apenas uma classe na base de dados de validação. A arquitetura *VGG16*, forneceu o melhor modelo, com resultados acima dos 90% de acurácia, realizando *fine tune* em todas as camadas. A partir deste resultado, geramos um novo modelo utilizando 100 mil imagens de treinamento com a *VGG16*, onde alcançamos nosso melhor resultado, 96% de acurácia no conjunto de validação e 97% de acurácia no conjunto de testes com aproximadamente 90 mil imagens.

A Figura 11 nos mostra o comportamento do custo da função de perda *binary cross entropy* em relação à quantidade de épocas utilizadas para a convergência do método nos conjuntos de treino e validação. Como pode ser observado, o custo da função alcança seu mínimo aproximado com cerca de 25 épocas para o conjunto de treino, enquanto no conjunto de validação esse mínimo é alcançado com cerca de 10 épocas.

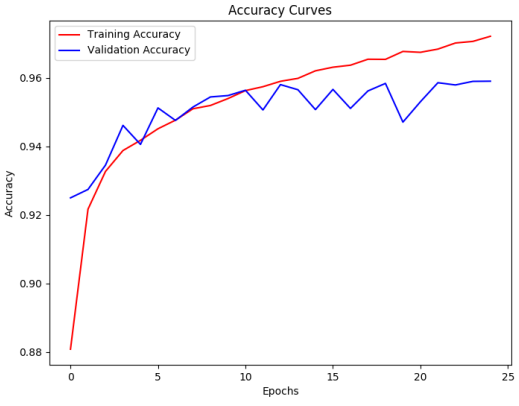


Figura 12. Comportamento da acurácia para o melhor modelo com as CNN's.

Na Figura 12, podemos ver o ganho de acurácia normalizada também para este melhor modelo encontrado. Pelo

Tabela II
COMPARAÇÃO DOS RESULTADOS DOS EXPERIMENTOS COM AS CNN'S

Arquitetura	Quantidade de imagens	Tamanho da imagem	Acurácia	Precisão		Recall		F1 score	
				0	1	0	1	0	1
CNN implementada	10000	128×128	0,8945	0,91	0,82	0,96	0,66	0,93	0,73
CNN implementada	10000	256×256	0,9072	0,91	0,87	0,97	0,68	0,94	0,76
CNN implementada	50000	128×128	0,8835	0,91	0,77	0,94	0,68	0,93	0,72
CNN implementada	50000	256×256	0,8942	0,91	0,85	0,97	0,67	0,93	0,74
SqueezeNet (classificação)	10000	128×128	0,7795	0,78	0	1	0	0,88	0
SqueezeNet (específica)	10000	128×128	0,7795	0,78	0	1	0	0,88	0
SqueezeNet (todas)	10000	128×128	0,7795	0,78	0	1	0	0,88	0
VGG16 (classificação)	10000	128×128	0,8335	0,94	0,59	0,84	0,8	0,89	0,68
VGG16 (específica)	10000	128×128	0,8880	0,94	0,72	0,91	0,81	0,93	0,76
VGG16 (todas)	10000	128×128	0,9235	0,97	0,79	0,93	0,9	0,95	0,84
ResNet-50 (classificação)	10000	128×128	0,7795	0,78	0	1	0	0,88	0
ResNet-50 (específica)	10000	128×128	0,2205	0,5	0,22	0	1	0	0,36
ResNet-50 (todas)	10000	128×128	0,7340	0,98	0,45	0,67	0,95	0,8	0,61
VGG16 (todas)	100000	128×128	0,959	0,97	0,92	0,98	0,89	0,97	0,91
TESTE	88486	128×128	0,9715	0,98	0,91	0,98	0,91	0,98	0,91

comportamento apresentado, podemos constatar que o ganho de acurácia é crescente em todo o domínio de épocas para o conjunto de treinamento, enquanto que para o conjunto de validação, esse ganho começa a oscilar a partir da quarta época, aproximadamente.

Figura 13. Comparação de imagens com navios detectados e não detectados com as CNN's

Na Figura 13, podemos ver os tipos de imagens que o modelo acerta, imagens mais limpas ou com poucas nuvens em que o navio se destaca, e os que não acertam, imagens com mais ruídos, cor de mar não padrão e navios bem pequenos que acabam sendo confundidos com ondas.

C. Detecção e Segmentação

Para a detecção e segmentação das imagens de satélite, organizamos os experimentos também em quatro etapas, porém não experimentamos diversas arquiteturas ou *frameworks* como realizado na seção anterior. Primeiramente, avaliamos as duas arquiteturas utilizadas no *backbone* do *framework Mask R-CNN*, fizemos então um *fine tune* em diversas camadas, em seguida, fizemos uma aumentação de dados, e por fim, utilizamos as imagens na resolução original. Em cada parte, trazemos os resultados obtidos das métricas apresentadas na metodologia e para nosso melhor modelo, apresentamos algumas imagens que nosso melhor modelo acertou e errou da base de dados para o teste.

1) ResNet-50 versus ResNet-101: Dentre os experimentos realizados, primeiro testamos os dois *backbones* existentes no *framework Mask R-CNN* e embora a rede *ResNet-101* seja mais profunda que a rede *ResNet-50*, foi a rede *ResNet-50* que atingiu melhor acurácia, cerca de 42% de acurácia, enquanto a *ResNet-101* apresentou aproximadamente 33% de acurácia. Mesmo assim este resultado está bem abaixo do esperado, sendo considerado insatisfatório. Nossa hipótese é que a quantidade de imagens para treinamento seja insuficiente para treinar um modelo com uma arquitetura com muitos parâmetros e profunda como estas.

2) Transfer Learning: Posteriormente, utilizando a *ResNet-50* no *backbone* do *framework*, experimentamos a técnica de transferência de aprendizado utilizando os pesos da base *MS COCO* treinada no *framework Mask R-CNN*. Tal base é muito utilizada na literatura para resolução de problemas de visão computacional tais como detecção e segmentação de objetos. Avaliamos a transferência de aprendizado em diferentes níveis desde um *fine tuning* desde as últimas camadas (heads) até um *fine tuning* com todas as camadas. Apenas com as camadas superiores, de classificação, obtemos um resultado muito ruim, apenas 30% de acurácia, aumentamos as camadas descongeladas para treinamento, atingindo as camadas que aprendem características mais específicas do objeto em questão, porém o resultado ainda permaneceu praticamente o mesmo, 32%. Fizemos então um *fine tune* em todas as camadas e obtivemos acurácia de 44% aproximadamente.

3) Data Augmentation: Para tentar melhorar os resultados obtidos fizemos aumentação dos dados gerando imagens rotacionadas, com *flip* e aplicando alguns filtros tais como Filtro Gaussiano, no entanto essa técnica não apresentou melhorias e o resultado permaneceu com baixa acurácia, cerca de 42%.

Observando algumas imagens da base de dados concluímos que muitas imagens possuíam navios que ocupam poucos *pixels* na imagem e como utilizamos imagens com tamanho reduzido, 256×256, esses navios acabam sendo perdidos na imagem e não sendo detectados pela rede. Dessa forma, treinamos a rede utilizando as imagens com a resolução original de 768×768 *pixels* e o resultado melhorou significativamente atingindo cerca de 56% de acertos.

Por fim, treinamos esse mesmo modelo que apresentou

melhor acurácia para todas as imagens da base de dados, com quase 200 mil imagens, incluindo as imagens que não possuem navios, e para nossa surpresa a acurácia praticamente se manteve, 57%, o que indica que as imagens que não possuem navio não contribuem com *features* para melhorar o desempenho da rede.

Ao final, aplicamos o melhor modelo encontrado no conjunto de teste e não obtivemos o resultado esperado, a acurácia do teste deu bem abaixo do nosso conjunto de validação, cerca de 43%.

Analisando os resultados obtidos pelo modelo treinado com o *framework Mask R-CNN*, observamos que a rede teve boa acurácia para casos onde os navios se destacam do fundo, com o mar em sua cor padrão por exemplo, como mostra a Figura 14. Porém em casos de imagens de encostas, portos ou de imagens ruidosas, Figura 15, a rede não apresentou bom desempenho detectando navios em objetos como prédios e faixas de terras.

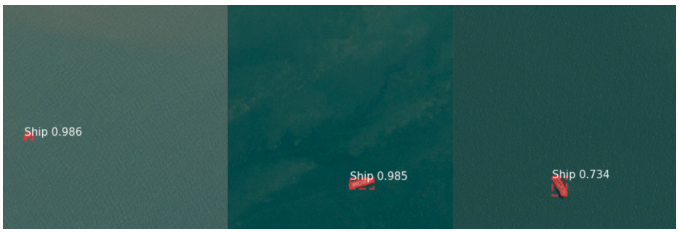


Figura 14. Imagens que a rede do *framework Mask R-CNN* acertou.

Figura 15. Imagens que a rede do *framework Mask R-CNN* errou.

V. Conclusões e Trabalhos Futuros

Neste projeto, foram propostas implementações de soluções explorando modelos generalistas, de forma a evitar *overfitting*, para classificação, detecção, extração e segmentação de posições de navios em oceanos em uma base de dados de imagens de satélite. Primeiramente, propusemos uma metodologia que consistia resumidamente em analisar os dados disponibilizados, simplificar o problema de detecção e segmentação para um problema de classificação apenas, extrair características desse conjunto de imagens, a fim de melhorar a acurácia dos modelos encontrados, para a partir daí exploramos técnicas de detecção e segmentação. Na *baseline*, avaliamos modelos de redes neurais que se adequassem melhor ao problema de classificação. Os experimentos foram divididos em dez partes, onde variamos a resolução das imagens, tipo dos dados - transformação (RGB, Escala de cinza, DCT, DFT) e a arquitetura utilizada em relação à resolução da imagem. Os resultados foram satisfatórios em relação ao erro e acurácia apresentados, e dentre os experimentos, o modelo com a imagem de resolução 128×128 em DCT apresentou os melhores resultados, com alta taxa de acertos. Além da *baseline*, foram explorados também técnicas de *deep learning*, como *CNN - Convolutional Neural Network* para obter maior acurácia em nossos resultados de classificação, e obter resultados para a detecção da posição de navios nas imagens de satélite. Para o problema de classificação, os experimentos foram executados em quatro etapas, onde primeiramente foi avaliada uma CNN muito similar à *LeNet*, em seguida, realizamos um *fine tune* na arquitetura *SqueezeNet*, depois, obtivemos modelos utilizando a arquitetura *VGG16*, e finalmente, experimentamos a arquitetura *ResNet-50*. Os resultados obtidos mostraram que a *VGG16* e a CNN implementada obtiveram os melhores modelos, apresentando métricas satisfatórias, o que não pode ser constatado nas outras arquiteturas testadas. Já no problema de detecção e segmentação de objetos por instâncias, realizamos experimentos em diversas etapas. Inicialmente, experimentamos as duas arquiteturas utilizadas no *backbone* do *framework Mask R-CNN*, fizemos então um *fine tune* em diversas camadas, em seguida, fizemos uma aumentação de dados, e por fim, utilizamos as imagens na resolução original. Os resultados não foram satisfatórios, obtivemos como melhor resultado metade dos acertos, aproximadamente, com as imagens em alta resolução. Nossa suspeita é que a quantidade de imagens, cerca de 42 mil, são insuficientes para treinarmos as profundas redes do *framework Mask R-CNN*, já que os pesos utilizados no *fine tune* não contribuíram para este problema em específico. Sugerimos, como trabalhos futuros, utilizar outras redes para classificação, como a *GoogLeNet*, apesar de termos conseguido resultados satisfatórios com a *VGG16*. Ainda, sugerimos, para a parte de detecção e segmentação, utilizar outros *frameworks*, como a *Mask R-CNN*, que tenham em seu *backbone* redes convolucionais diferentes da *ResNet* (50 ou 101), como a *VGG16* ou *19* para extração de *features* das imagens e detecção de objetos. Além disso, sugerimos também fazer um *Ensemble* com essas redes e podemos acrescentar ao modelo uma etapa de classificação e detecção de regiões de mares e oceanos removendo assim, as áreas de encostas, evitando a falsa detecção de navios nessas áreas, a fim de melhorar os resultados obtidos.

Referências

[1] Base de dados da *Airbus*, link: https://www.kaggle.com/c/airbus-ship-detection/data

[2] Aurelien Geron. "Hands-On Machine Learning with Scikit-Learn and TensorFlow". O'Reilly, 2017.

[3] *Mask R-CNN explained*, link: https://becominghuman.ai/mask-r-cnn-explained-7f82bec890e3

[4] *Mask R-CNN*, link: https://arxiv.org/pdf/1703.06870.pdf

[5] *Mask R-CNN framework*, link: https://github.com/matterport/Mask_RCNN

[6] Metrics for Object Detection, link: https://github.com/rafaelpadilla/Object-Detection-Metrics