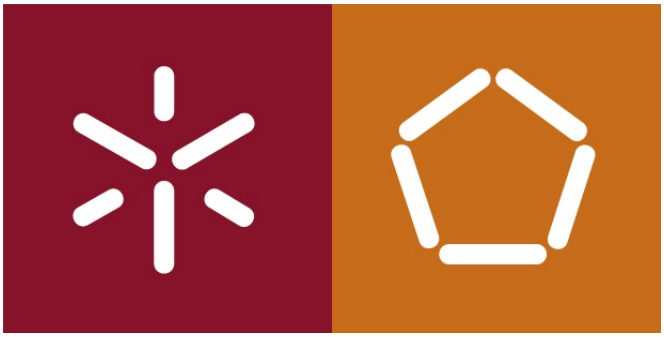


Universidade do Minho
Departamento de Informática

Laboratório em Engenharia Informática

Deteção de Alterações no Solo Português através de Sequências de Imagens de Satélite

Gonçalo Almeida, A84610
Pedro Oliveira, A83762
Luís Ferreira, A76936

22 de junho de 2021

Conteúdo

1 Introdução **3**
- 1.1 Contexto . . . 3
  - 1.1.1 Sensores SAR . . . 3
  - 1.1.2 Sentinel-1 . . . 3

2 Preparação dos Dados **4**
- 2.1 Conjunto de Dados . . . 4
- 2.2 Manipulação . . . 4
  - 2.2.1 Conversão dos Dados . . . 4
  - 2.2.2 Agrupamento por Pares . . . 5
  - 2.2.3 Não Supervisionado para Supervisionado . . . 5
  - 2.2.4 Redução de Dimensionalidade . . . 5
  - 2.2.5 Normalização . . . 6
  - 2.2.6 Divisão em Conjuntos de Treino, Validação e Teste . . . 6

3 Modelo U-Net **7**
- 3.1 Arquitetura . . . 7
- 3.2 Treino do Modelo . . . 7
- 3.3 Análise de Resultados . . . 8
  - 3.3.1 *Clustering* . . . 10

4 Conclusões **12**

A Anexos **14**
- A.1 Arquitetura do Modelo U-Net . . . 14

1 Introdução

O presente trabalho prático foi desenvolvido no âmbito da unidade curricular Laboratório em Engenharia Informática do curso MIEI disponibilizado pela Universidade do Minho.

A deteção de alterações é um processo de análise da evolução do ambiente ou da construção de estruturas humanas de uma certa área que ajuda na tomada de decisões sobre a mesma. Com este projeto pretende-se identificar alterações no solo Português através de sequências de imagens satélite da mesma localização obtidas em instantes temporais distintos. Para tal, foram obtidas imagens de uma determinada área de interesse escolhida pelo grupo e foi implementado um modelo de Deep Learning (DL) capaz de detetar as alterações temporais, o que implicou a manipulação dos dados das imagens para serem alimentados a tal modelo. Por fim, foi realizada uma análise dos resultados obtidos com a nossa abordagem.

1.1 Contexto

1.1.1 Sensores SAR

Os sensores do tipo *Synthetic Apperture Radar* (SAR) são adequados para aplicações de deteção de alterações no solo visto que apresentam vantagens como o facto de funcionarem durante o dia e a noite, com e sem cobertura de núvens.

Este tipo de sensores são montados em plataformas em movimento, como aeronaves ou satélites, e emitem pulsos sequenciais de ondas rádio sobre uma região alvo. A distância que o SAR viaja entre a emissão do sinal e a receção do seu eco confere-lhe a sua propriedade de abertura sintética de antena que, após o processamento dos vários sinais recolhidos, permite formar uma imagem de alta resolução recorrendo a antenas de relativa fraca capacidade.

A versatilidade e rigor deste tipo de imagens tornam-nas extremamente úteis em diferentes contextos, tais como topologia de grandes áreas, controlo de desflorestamento, planeamento urbano, deteção de derrames petrolíferos e, cheias assim como várias aplicações militares.

1.1.2 Sentinel-1

A missão Sentinel-1 faz parte da iniciativa Cupernicus das instituições European Commission e European Space Agency e é realizada por uma constelação de dois satélites que utilizam sensores do tipo SAR para a observação da Terra através de imagens obtidas numa grande amplitude de condições atmosféricas. Os satélites denominados Sentinel-1A e Sentinel-1B, lançados em 2014 e 2016 respetivamente, trabalham com imagens de banda C e operam em quatro modos exclusivos de aquisição [1], sendo o modo *Interferometric Wide swath* (IW) o *default* para a aquisição de dados em terra e aquele que utilizamos.

2 Preparação dos Dados

Nesta secção iremos descrever todos os processos realizados para obter os dados necessários para a análise pretendida, bem como a manipulação dos mesmos de forma a poderem ser alimentados a um modelo DL de modo a serem obtidos bons resultados.

2.1 Conjunto de Dados

O conjunto de dados com os quais foram realizadas todas as experiências é proveniente da coleção Sentinel-1 SAR GRD [2] disponível na plataforma Google Earth Engine (GEE). Sendo que esta coleção contêm imagens de áreas por todo o planeta, definimos uma área de interesse referente à cidade de Braga e filtramos as imagens de modo a obter apenas aquelas que incluem tal área. Para que as imagens sejam comparadas da melhor forma possível apenas consideramos a passagem orbital *Ascending*, o que implica que todas as capturas tenham sido realizadas com o mesmo ângulo de incidência do sensor. Com isto, a diferença temporal entre cada captura é de 12 dias. Por fim, ordenamos os dados obtidos por data de captura.

2.2 Manipulação

Após ter sido obtida a coleção de imagens da API do GEE começamos por cortar as mesmas de forma a apenas trabalhar com os dados da área de interesse definida.

2.2.1 Conversão dos Dados

Visto que estas imagens são dados em banda C com dupla polarização tivemos que os converter para uma outra estrutura com maior eficiência e liberdade de manipulação. Com esta dupla polarização temos disponíveis as seguintes quatro bandas de polarização:

- VV: co-polarização única, transmissão/recepção vertical
- HH: co-polarização única, transmissão/recepção horizontal
- VV + VH: polarização cruzada de dupla banda, transmissão vertical e recepção horizontal
- HH + HV: polarização cruzada de dupla banda, transmissão horizontal e recepção vertical

Começamos por converter cada imagem GEE para uma imagem Pillow, proveniente do *package* com o mesmo nome, através de um composto RGB com as bandas que, para dados da Sentinel-1, a prática mais comum é a banda VV para vermelho, a banda HH para verde e a razão entre as bandas VV e HH para azul.

De seguida, garantimos que todas as imagens resultantes têm a mesma resolução (1024x1024) e convertemos as mesmas para Numpy *Arrays*, em que o valor de cada píxel é representado em *grayscale*.

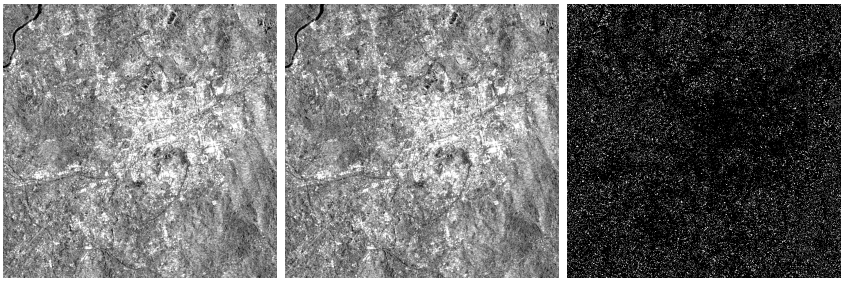
Por fim, exportamos os dados para um ficheiro de forma a que todo este demoroso processo não seja realizado sempre que queremos realizar testes.

2.2.2 Agrupamento por Pares

Para detetar as alterações ao longo do tempo consideramos pares de imagens capturadas consecutivamente, isto é, agrupamos cada imagem num par com a imagem anteriormente obtida e num outro par com a imagem posteriormente obtida. Com isto, para N imagens no conjunto de dados, temos $N - 1$ pares.

2.2.3 Não Supervisionado para Supervisionado

Para tornar o problema supervisionado de modo a que o modelo DL possa aprender a prever padrões do *output* dos dados de *input*, consideramos a diferença dos valores dos píxeis entre as duas imagens de cada par, tendo em conta um *threshold* para ignorar diferenças mínimas. Com isto, consideramos os valores superiores ao *threshold* como "houve mudanças"e atribuímos o valor 1 e, por outro lado, os valores inferiores como "não houve mudanças"e atribuímos o valor 0, tendo assim um problema binário.

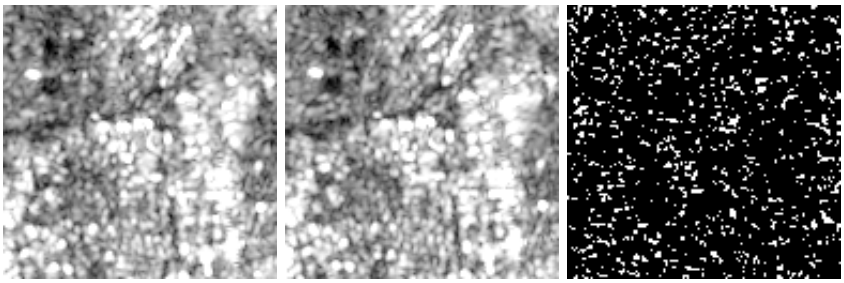


(a) (b) (c)

Figura 1: (a) Imagem capturada no instante $t1$ (b) Imagem capturada no instante $t2$ (c) Alterações detetadas entre os dois instantes

2.2.4 Redução de Dimensionalidade

Visto que temos imagens com a dimensão 1024x1024 decidimos separá-las em imagens mais pequenas com a dimensão 128x128. Desta forma, não só diminuímos o tamanho da rede neuronal do modelo necessário, como a carga computacional requerida para o treinar, e também aumentamos o número de dados disponíveis com os quais o modelo poderá ser treinado.



(a) (b) (c)

Figura 2: (a) Secção da imagem capturada no instante $t1$ (b) Secção da imagem capturada no instante $t2$ (c) Secção das alterações detetadas entre os dois instantes

2.2.5 Normalização

Quando uma rede neuronal é alimentada com dados não escalados e a dimensão dos dados de *input* for suficientemente grande, os processos de aprendizagem e convergência podem ser afetados. Para evitar este problema normalizamos os dados das imagens, escalando os seus valores para o intervalo [0, 1].

2.2.6 Divisão em Conjuntos de Treino, Validação e Teste

Por fim, separamos os dados em conjuntos de treino e de teste para estimar a performance dos modelos considerando 90% dos dados para treino e os restantes 10% para teste. No entanto, 20% dos dados de treino foram considerados para validação, sendo que este conjunto fornece uma estimativa de quão bem o modelo está a ser treinado durante o ajuste dos seus hiperparâmetros.

3 Modelo U-Net

Nesta secção iremos descrever a arquitetura do modelo escolhido para a deteção das alterações, bem como o processo de treino do próprio modelo e uma análise dos resultados obtidos.

3.1 Arquitetura

Para a implementação de um modelo DL consideramos uma arquitetura U-Net. Esta arquitetura encontra-se dividida em duas partes distintas, uma rede constituída por blocos de contração denominada *encoder* e uma rede constituída por blocos de expansão denominada *decoder*. O *encoder* tem como objetivo reduzir o espaço dimensional dos dados em cada bloco de forma a capturar o contexto das imagens de *input* para uma representação das suas *features*. Por outro lado, o *decoder* tem como objetivo projetar as *features* codificadas pelo *encoder* para um espaço de maior dimensão. Existe ainda um bloco entre estas duas redes denominado *bottleneck*. Por fim, uma última camada convolucional está encarregue de fazer a predição de cada píxel da imagem de *output*.

Para a nossa abordagem, cada bloco de contração é constituído pela sequência de duas camadas convolucionais e uma camada de *Max-Pooling*. Por outro lado, cada bloco de expansão é constituído pela sequência de uma camada *Up-Sampling*, uma camada de concatenação e duas camadas convolucionais. O *bottleneck* é constituído por duas camadas convolucionais. As função de ativação de todas as camadas convolucionais destas secções da rede neuronal é a função *ReLu*. Tratando-se de um problema binário, a função da última camada convolucional é a função *Sigmoid*. Como esta arquitetura não contêm quaisquer camada *Dense* em que temos que definir o número de neurónios para um tamanho de imagem específico, uma das vantagens que apresenta é o facto de poder aceitar uma variedade de tamanhos das imagens de *input*.

A arquitetura do modelo implementada é descrita graficamente no anexo A.1.

3.2 Treino do Modelo

A segmentação de imagens não é nada mais do que um problema de classificação, em que temos uma *label* para cada píxel em vez de uma *label* para a imagem em si. Contudo, no nosso caso, temos um problema desbalanceado, isto é, a quantidade de píxeis classificados como 0 (não houve alterações) é consideravelmente superior à quantidade de píxeis classificados como 1 (houve alterações). Assim sendo, devemos utilizar métricas adequadas para avaliar a performance do modelo. As métricas escolhidas foram a *Precision* que indica a percentagem de valores corretamente classificados como positivos (valor 1) e o *Recall* que indica a percentagem de valores verdadeiramente positivos corretamente classificados.

Sendo esta uma segmentação binária, como função de custo consideramos a *Binary Cross Entropy* pois é compatível com problemas de multiplas classificações ao mesmo tempo. Consideramos também a *Dice Loss* que é baseada no

coeficiente de *Dice*, originalmente desenvolvido para dados binários, que essencialmente mede a sobreposição de duas amostras. Esta medida varia de 0 a 1, em que 1 denota uma sobreposição perfeita.

Por fim, em termos de *optimizer*, consideramos o *Adam*.

Após compilar do modelo passamos então ao seu treino. De forma a encontrar os melhores hiperparâmetros seguimos a abordagem *grid search* em que geramos múltiplas configurações dos hiperparâmetros, podendo assim comparar os resultados de cada configuração. Contudo, por limitações de recursos, não foi possível implementar este processo de uma forma automática.

3.3 Análise de Resultados

Para as diferentes configurações escolhemos variar os valores de três hiperparâmetros, a taxa de aprendizagem do *optimizer Adam* (α), o número de épocas (*epochs*) e o tamanho de cada *batch* (*batch size*). Contudo, para os testes realizados, mantivemos o valor da taxa de aprendizagem pois, para o modelo e dados em causa, com diferentes valores eram obtidos resultados fracos ou inconsistentes. De qualquer forma, a nossa abordagem permite alterar este parâmetro facilmente.

Nas seguintes tabelas são apresentados os valores da perda, *Precision* e *Recall* para as duas funções de custo referidas e configurações geradas. Estes resultados foram obtidos avaliando o modelo com o conjunto de dados de teste.

α	*Epochs*	*Batch Size*	*Loss*	*Precision*	*Recall*
0.001	5	32	0.02307	0.99478	0.92318
		64	0.05673	0.94022	0.88589
	10	32	0.01169	0.99718	0.96312
		64	0.01964	0.98859	0.95137
	15	32	0.00750	0.99570	0.98975
		64	0.01383	0.97343	0.99038

Tabela 1: Resultados obtidos para os dados de teste com *Binary Cross Entropy*

α	*Epochs*	*Batch Size*	*Loss*	*Precision*	*Recall*
0.001	5	32	0.02126	0.97115	0.98574
		64	0.04563	0.94395	0.95378
	10	32	0.01689	0.99780	0.95985
		64	0.02082	0.97179	0.98720
	15	32	0.01216	0.99848	0.97235
		64	0.01893	0.99269	0.96389

Tabela 2: Resultados obtidos para os dados de teste com *Dice*

Com base nos resultados apresentados, pode-se observar que os resultados, para todas as configurações, foram bastante positivos, embora uns mais que outros. Os valores da *Precision*, nos melhores casos, apresentam valores de 98% e 99% o que indica que, dos píxeis que o modelo classificou como alterações, quase todos foram corretamente classificados. Por outro lado, os valores do *Recall* também apresentam valores elevados, o que indica que o modelo classificou corretamente quase todos os píxeis que verdadeiramente correspondem a alterações.

Os modelos com uma arquitetura U-Net são notórios de apresentarem bons resultados para problemas de segmentação, especialmente quando se trata de uma segmentação binária, como é o nosso caso.

Como foi referido anteriormente, a função de ativação da última camada do modelo é a função *Sigmoid*, o que implica que para cada píxel temos um valor entre 0 e 1. Sendo que pretendemos obter apenas valores binários, classificamos os píxeis com valor abaixo de um determinado *threshold* como 0 e com valor acima como 1. O valor do *threshold* que utilizamos e com o qual obtivemos bons resultados foi o valor 0.5, sendo este o valor ideal pois é o valor médio entre os dois extremos, o que reforça a boa performance do modelo.

Os resultados podem ser analisados de uma forma mais prática através da comparação visual entre as alterações reais e as alterações previstas pelo modelo.

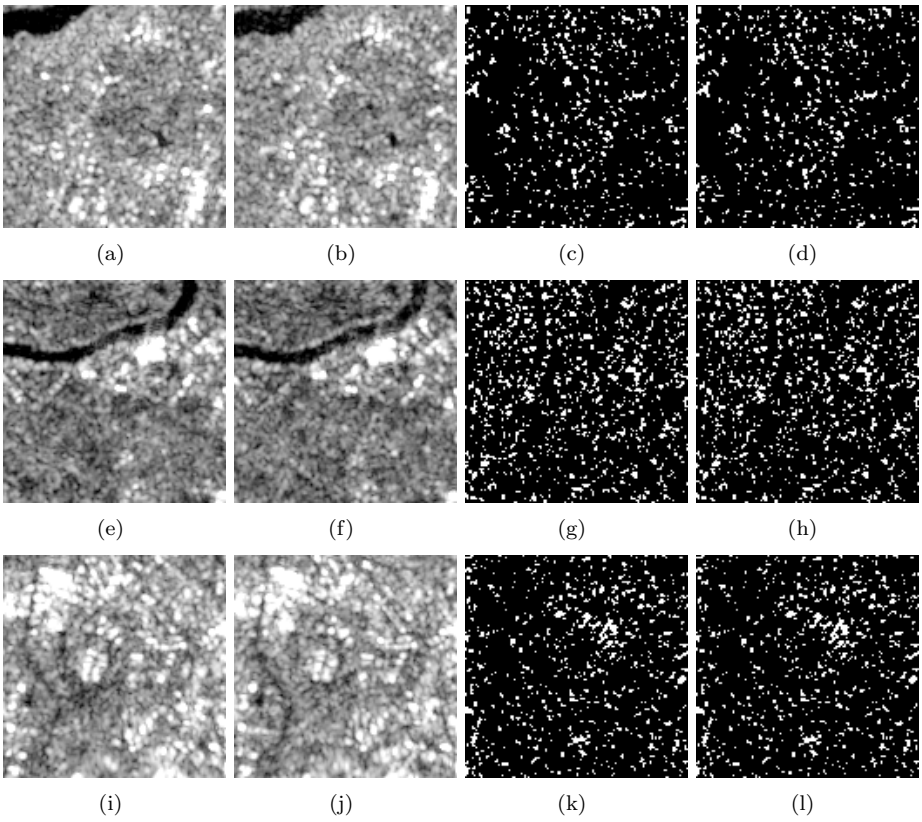


Figura 3: (a, e, i) Secção da imagem capturada no instante $t1$ (b, f, j) Secção da imagem capturada no instante $t2$ (c, g, k) Secção das alterações detetadas entre os dois instantes (d, h, l) Predição das alterações realizada pelo modelo

Com estas predições confirmam-se então os valores obtidos para as métricas avaliadas, visto que as predições aproximam-se detalhadamente aos dados reais.

3.3.1 *Clustering*

Por fim, com o intuito de desprezar a deteção de alterações mínimas, foi aplicado um algoritmo de *clustering DBScan* [3] às previsões do modelo. Idealmente, este processo seria realizado na fase do tratamento dos dados para que o modelo não tivesse que prever todas as alterações. Contudo, devido às limitações dos recursos a nós disponíveis, a carga computacional deste processo não permitiu a filtragem de todas as imagens do conjunto de dados. Alternativamente, seria ideal a escolha de um outro algoritmo que não implicasse tantos recursos computacionais. De qualquer forma, como o modelo não apresentou uma deficiência na qualidade de resultados, mesmo com todas as alterações, a aplicação do *clustering* antes dos dados serem alimentados ao modelo não se mostrou necessária.

De seguida são apresentados alguns exemplos da aplicação deste método às predições das alterações feitas pelo modelo.

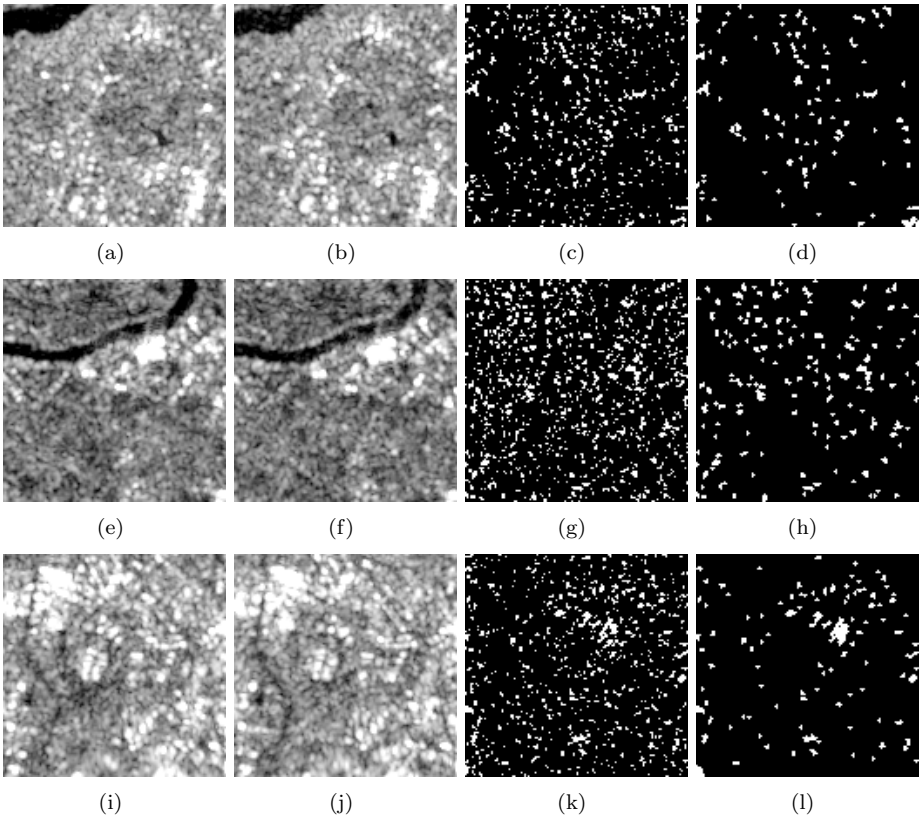


Figura 4: (a, e, i) Secção da imagem capturada no instante $t1$ (b, f, j) Secção da imagem capturada no instante $t2$ (c, g, k) Predição das alterações realizada pelo modelo (d, h, l) Predição das alterações após *clustering*

4 Conclusões

Uma vez terminada a exposição do trabalho efetuado podemos então tirar algumas conclusões sobre o mesmo.

Como ponto inicial começamos por realçar a importância das reuniões realizadas ao longo do semestre com o coordenador deste projeto, visto que permitiu esclarecer algumas dúvidas que foram surgindo e, também foram-nos sugeridas algumas abordagens que serviram para uma melhor compreensão do contexto deste trabalho.

Foi fundamental o estudo do tipo de dados fornecidos pela Sentinel-1, visto que se tratam de imagens capturadas através de sensores SAR e não através dos sensores ópticos mais comuns. Este estudo foi facilitado com a análise dos tutoriais [4] fornecidos pela Google sobre a deteção de alterações em imagens SAR.

Incidindo agora sobre o modelo DL, o maior desafio encontrado ao longo do desenvolvimento deste projeto foi a escolha da arquitetura mas, após o estudo das abordagens mais utilizadas para a deteção de alterações [5] [6], a arquitetura U-Net mostrou-se uma escolha sólida em termos de resultados. Um outro obstáculo foi o facto de ter sido mal implementada uma transformação matricial na manipulação dos dados, o que resultou em resultados anormais pois o modelo apenas previa as alterações ao longo de um padrão diagonal na imagem. Felizmente, fomos capazes de detetar a causa deste problema e corrigi-lo. Isto reforça a grande importância e necessidade de um bom tratamento de dados para serem obtidos bons resultados com modelos DL. Por fim, referir a importância de saber escolher as métricas para avaliar corretamente os modelo conforme o problema em causa.

Concluindo a exposição, consideramos que atingimos com sucesso os resultados pretendidos para este projeto.

Referências

[1] The European Space Agency. Acquisition Modes.
Disponível em:
https://sentinels.copernicus.eu/web/sentinel/user-guides/sentinel-1-sar/acquisition-modes [Acedido a 17 de Junho 2021]

[2] Google. Sentinel-1 SAR GRD: C-band Synthetic Aperture Radar Ground Range Detected, log scaling.
Disponível em:
https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS_S1_GRD [Acedido a 17 de Junho 2021]

[3] Scikit-learn. Clustering algorithm.
Disponível em:
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html [Acedido a 17 Junho 2021]

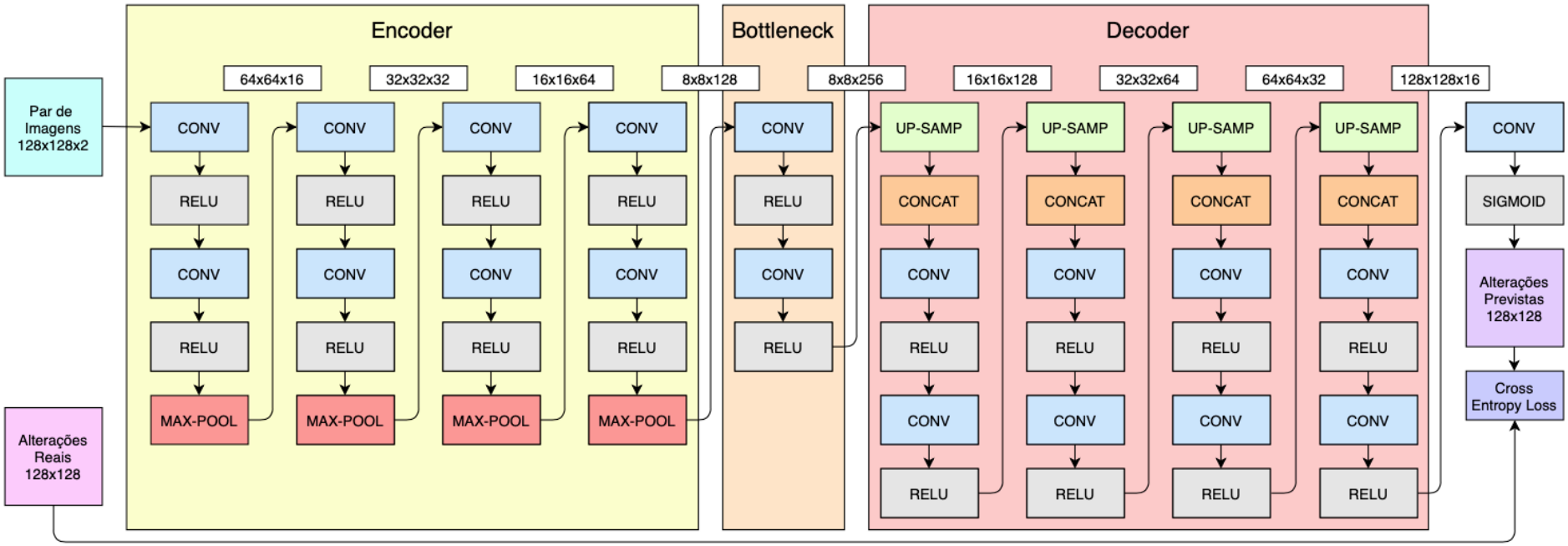
[4] Mort Canty, Google. Detecting Changes in Sentinel-1 Imagery (Part 1).
Disponível em:
https://developers.google.com/earth-engine/tutorials/community/detecting-changes-in-sentinel-1-imagery-pt-1 [Acedido a 17 Junho 2021]

[5] Kevin Louis de Jong, Anna Sergeevna Bosman. Unsupervised Change Detection in Satellite Images Using Convolutional Neural Networks.

[6] Pablo F. Alcantarilla, Simon Stent, German Ros, Roberto Arroyo, Riccardo Gherardi. Street-View Change Detection with Deconvolutional Networks.

A Anexos

A.1 Arquitetura do Modelo U-Net


Encoder
Bottleneck
Decoder
Par de Imagens 128x128x2
Alterações Reais 128x128
64x64x16
32x32x32
16x16x64
8x8x128
8x8x256
16x16x128
32x32x64
64x64x32
128x128x16
CONV
RELU
MAX-POOL
UP-SAMP
CONCAT
SIGMOID
Alterações Previstas 128x128
Cross Entropy Loss