NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE INFORMÁTICA

MARTA FILIPA DA CUNHA PAZ

Licenciatura em Ciências e Engenharia Informática

SEGMENTAÇÃO NÃO SUPERVISIONADA EM MICROSCOPIA DE BACTÉRIAS

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
Novembro, 2021

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE INFORMÁTICA

SEGMENTAÇÃO NÃO SUPERVISIONADA EM MICROSCOPIA DE BACTÉRIAS

MARTA FILIPA DA CUNHA PAZ

Licenciatura em Ciências e Engenharia Informática

Orientador: Ludwig Krippahl
Professor Assistente, Universidade Nova de Lisboa

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
Novembro, 2021

Segmentação Não Supervisionada em Microscopia de Bactérias

Este documento foi gerado com o processador (pdf/Xe/Lua)LaTeXe o modelo NOVAthesis (v6.7.0) [27].

Agradecimentos

A realização desta dissertação contou com o apoio e incentivo de diversas pessoas que sem os quais não teria sido possível completá-la.

Ao meu orientador, professor Ludwig Krippahl, pela sua orientação, pelo apoio ao longo de todo o desenvolvimento desta dissertação, disponibilidade, por toda as dúvidas esclarecidas, por tudo o que me ensinou e por todo o incentivo que me forneceu para finalizá-la.

A todos os docentes do Mestrado Integrado em Engenharia Informática por todo o conhecimento que me transmitiram ao longo dos cinco anos de curso e que foram fundamentais para o desenvolvimento deste trabalho.

Aos meus pais e família por todo o apoio incondicional que me deram, por toda a paciência que tiveram comigo ao longo não só da tese, mas dos 5 anos de curso, por toda a ajuda na superação de todos os obstáculos. Mas acima de tudo por me apoiarem em todas as decisões que tomei até ao momento e me incentivarem a ir sempre mais longe.

A todos os meus amigos que conheci não só ao longo destes 5 anos de faculdade, mas todos os que estiveram e continuam sempre presentes, que me apoiaram sempre, me incentivaram a ir mais longe e pela companhia nas alturas de mais trabalho.

“Have the courage to follow your heart and intuition. They somehow know what you truly want to become.” (Steve jobs)

Resumo

A segmentação de imagens de microscopia de bactérias é extremamente importante para a obtenção de estatísticas de populações de bactérias. Apesar dos resultados razoáveis apresentados pelas típicas técnicas de visão computacional, estas apresentam algumas limitações, principalmente no que toca a grupos de bactérias juntas. Nestes casos, os algoritmos utilizados cometem erros ao segmentar duas ou mais bactérias desses grupos como sendo uma só. A aprendizagem supervisionada profunda tem a capacidade de melhorar esses resultados. Contudo, requer uma grande quantidade de imagens manualmente selecionados para serem utilizadas durante o processo de treino dos modelos. No caso de imagens de microscopia de bactérias esse processo de seleção requer a anotação manual de cada bactéria presente em cada imagem, o que demora bastante tempo. Desta forma, encontrar uma solução para a segmentação não supervisionada de imagens de microscopia de bactérias iria não só resolver o problema da necessidade de dados rotulados, mas também o problema de grupos de bactérias juntas.

A arquitetura W-Net funciona como um autoencoder e utiliza camadas totalmente convolucionais para aprender a criar uma representação da imagem de input ao mesmo tempo que produz uma máscara de segmentação da mesma. Durante esse processo é realizada a minimização da função de perda de reconstrução do autoencoder e da função de perda do corte normalizado suave da camada de codificação. Após este processo, são aplicados dois passos de pós-processamento de forma a obter a segmentação final da imagem fornecida. Esta arquitetura será adaptada de forma a acomodar uma variação no número de filtros, no número de camadas utilizadas e nos passos de pós-processamento.

A solução apresentada tem como objetivo otimizar a rede W-Net e testar se é possível utilizar essa rede para segmentação de imagens de microscopia de bactérias de forma a resolver ambos os problemas mencionados anteriormente: os erros ocorridos durante o processo de segmentação de grupos de bactérias juntas e a necessidade de grandes quantidades de dados manualmente rotulados.

Palavras-chave: Segmentação Não Supervisionada; W-Net; Imagens de Microscopia de Bactérias; Redes Neurais Convolucionais; Autoencoder Convolucional

Abstract

Segmentation of bacterial microscopy images is extremely important for obtaining statistics of bacterial populations. Despite de reasonable results presented by typical computer vision techniques, it presents some limitations, especially regarding groups of bacteria together. In such cases, the algorithms used make mistakes when targeting two or more bacteria from these groups as being just one. Deep supervised learning has the ability to improve these results. However, it requires many images manually selected to be used during the training process of the models. In the case of bacterial microscopy images, this selection process requires the manual annotation of each bacterium present in each image, which takes a long time. In this way, finding a solution for unsupervised segmentation of bacterial microscopy images would not only solve the problem of the need for labeled data, but also the problem of groups of bacteria together.

The W-net architecture works as an autoencoder and uses fully convolutional layers to learn how to create a representation of the input image while producing a segmentation mask for it. During this process, the reconstruction error of the autoencoder and the soft normalized cut loss of the encoder network are both minimized. After this process, two postprocessing steps are applied in order to obtain the final segmentation of the image provided. This architecture will be adapted to accommodate a variation in the number of filters, the number of layers used and in the postprocessing steps.

The presented solution aims to optimize the W-Net network and test whether it is possible to use this network for segmentation of bacteria microscopy image in order to solve both problems mention above: the error that occurred during the segmentation process of groups of bacteria together and the need for large amounts of manually labeled data.

Keywords: Unsupervised Segmentation; W-Net; Bacteria Microscopy Images; Convolutional Neural Networks; Convolutional Autoencoder

Índice

Índice de Figuras

Índice de Tabelas

1

Introdução

1.1 Motivação

A segmentação de imagens caracteriza-se por dividir uma imagem em vários segmentos, cada um contendo um objeto da imagem. A segmentação de imagens de bactérias é bastante importante para a obtenção de estatísticas sobre populações de bactérias, como o número de bactérias dessa população. Redes neurais convolucionais são utilizadas para este tipo de problemas devido à sua capacidade de aprender a partir de padrões, características ou texturas em vez de se basear nas variações de cor.

Técnicas recentes de visão computacional tiveram bons resultados com este tipo de imagens, no entanto, têm algumas limitações, especialmente no que toca a grupos de células juntas, como os grupos presentes na Figura 1.1. Nestes casos, o algoritmo não consegue distinguir com exatidão as células o que leva a, por vezes, segmentar duas ou mais células como se fossem apenas uma.

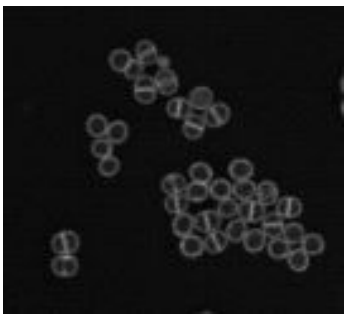


Figura 1.1: Grupos de Bactérias

A aprendizagem supervisionada profunda é um método de aprendizagem automática que utiliza uma ou mais camadas escondidas de forma a aprender gradualmente a partir das imagens previamente rotuladas dadas como input. Este método é uma possível solução do problema anteriormente descrito. Contundo, requer um grande número de imagens manualmente selecionadas para treinar os modelos. Essa seleção manual das imagens, para o problema em questão, consiste na marcação individual de cada bactéria presente em cada imagem, o que é um processo bastante demorado.

Outra forma de resolver o problema da segmentação de imagens é a utilização de aprendizagem não supervisionada em que as imagens fornecidas como input não têm qualquer rótulo. Desta forma, a aprendizagem baseia-se na descoberta de semelhanças entre as imagens dadas como input.

Os autoencoders são um tipo de aprendizagem não supervisionada baseados no processo de codificação descodificação. Entre todos os tipos existentes, os autoencoders convolucionais são os que se destacam na segmentação de imagens devido à sua capacidade de aprender a partir de padrões, características ou texturas em vez de se basear nas variações de cor. Estes autoencoders são treinados para reproduzir a imagem de input na camada de output. Desta forma, a utilização dos autoencoders convolucionais teve como objetivo eliminar a necessidade de se utilizar imagens manualmente selecionados para treinar os modelos e tentar reduzir o erro associado à segmentação de grupos de bactérias juntas.

1.2 Contexto

1.2.1 Aprendizagem Não Supervisionada versus Aprendizagem Supervisionada

Em ambos os tipos de aprendizagem [14], é fornecido um conjunto de dados que contém um conjunto de exemplos. Estes, por sua vez, são conjuntos de features que foram medidos quantitativamente a partir de um objeto ou evento que é suposto que o sistema aprenda.

Na aprendizagem não supervisionada são aprendidas propriedades úteis do conjunto de dados fornecido. No contexto de aprendizagem profunda, é usual querer-se aprender toda a distribuição de densidade que gera o conjunto de dados, seja explicitamente como na estimação de densidade ou implicitamente para tarefas como síntese ou eliminação de ruído. Existem outros algoritmos de aprendizagem não supervisionada que têm outras funções como é o caso do clustering que consiste na divisão do conjunto de dados em diferentes clusters com exemplos similares.

Na aprendizagem supervisionada, cada exemplo do conjunto de dados fornecido contém um rótulo e os algoritmos aprendem a discriminar os dados fornecidos com base nos diferentes rótulos existentes do conjunto de dados.

Por outras palavras [14], a aprendizagem não supervisionada envolve a observação de vários exemplos de um vetor aleatório x e tenta aprender, implicitamente ou explicitamente, a distribuição de probabilidades p(x), ou outras propriedades interessantes dessa distribuição. Enquanto a aprendizagem supervisionada envolve a observação de vários exemplos de um vetor aleatório x e um valor associado ou o vetor y, e aprende a prever y a partir de x, geralmente estimando p(y|x).

Resumidamente, na aprendizagem não supervisionada, os algoritmos recebem o conjunto de dados sem qualquer rótulo e aprendem de forma que os dados fornecidos façam sentido. Pelo contrário, na aprendizagem supervisionada, os algoritmos aprendem com

base nos rótulos dos dados fornecidos, conseguindo, no final, discriminar dados com base nesses mesmos rótulos.

1.2.2 Redes Neurais

Uma rede neural [38] é um conjunto de redes conectadas constituída por uma camada de input e outra de output. Entre essas duas, podem existir camadas escondidas. Este tipo de rede consegue resolver problemas quer estes sejam linearmente separáveis ou não.

Ligação input-camada escondida

Numa rede neural, o vetor de input x está conectado a cada nó da rede através de pesos w, que podem ser vistos como sendo vários vetores de peso formando uma matriz W. O número de colunas que constituem W é igual ao total de nós que a camada tem e o número de linhas que constituem W é igual ao total de features pertencentes a x. Assim, o output da camada escondida pode ser visto como o vetor 1.1.

$$h = z(w^T x + b) \tag{1.1}$$

Onde b é o vetor de bias cujos elementos correspondem a um nó e o tamanho de h é proporcional ao número de unidades escondidas.

Ligação camada escondida - camada escondida

Numa rede neural, é possível ter mais do que uma camada escondida. Neste caso, a matriz W pode ser expressa como uma matriz tridimensional em que terá tantos elementos na terceira dimensão e tantas camadas escondidas quanto a rede tem. Por conveniência, a i-ésima camada é referida por W_i. Assim, podemos referir ao output da i-ésima camada escondida como 1.2.

$$h_i = z(W_i^T h_{i-1} + b_i) \tag{1.2}$$

Para i = 2, 3, ..., k-1, onde k é o número total de camadas e h_1 é calculado com a equação da primeira camada que utiliza diretamente o x e não vai até à última camada porque h_k é calculado como descrito na secção seguinte.

Ligação camada escondida - output

O output geral da rede é o output da última camada e é dado por 1.3.

$$h_k = z(W_k^T h_{k-1} + b_k) \tag{1.3}$$

A última função de ativação é tipicamente diferente das ativações das camadas escondidas pois costuma depender do tipo de problema. Por exemplo, no caso de se estar a resolver uma regressão linear dever-se-á utilizar uma função linear. Já no caso de problemas de classificação, dever-se-á utilizar a ativação sigmoid.

Neste caso, apesar da aprendizagem ser em termos dos erros que a rede comete, os ajustamentos não poderão ser feitos diretamente nos dados que estão a ser classificados e previstos incorretamente, como acontece em muitos outros algoritmos de aprendizagem, como é o caso do perceptrão. Isto acontece porque os nós da última camada dependem da camada anterior e os dessa camada dependem da anterior e assim por diante. Deste modo, os ajustamentos de W e b terão de ser realizados de modo diferente para cada nó.

Uma forma de resolver esse problema seria aplicar técnicas de gradiente descendente na rede neural. Em geral, um algoritmo de gradiente descendente utiliza a derivada de uma função para encontrar o valor máximo (ou mínimo) que se pode ter para o conjunto de parâmetros que se está a derivar. Esse valor corresponde, tipicamente, ao zero da derivada da função.

A função que se pode utilizar é chamada de função de perda e é, usualmente, diferenciável de modo que seja possível aplicar o algoritmo de gradiente descendente. A função de perda pode ser definida pela Equação 1.4.

$$L = \frac{1}{N} \sum_{i=1}^{N} (y_i - h_{i,k})^2 \tag{1.4}$$

Esta função de perda é conhecida como erro quadrático médio (MSE): mede a diferença entre o valor real e o valor na camada de output h_k. Essa diferença é calculada em termos do quadrado e média dos seus elementos. Esta é uma boa função de perda pois é diferenciável e fácil de calcular. No entanto, existem muitas outras funções de perda que poderão ser utilizadas, dependendo do problema em questão.

Resumidamente [41], uma rede neural recebe dados como input num único vetor. O input passa por uma série de camadas escondidas em que cada uma consiste num conjunto de nós, sendo que cada um dos nós é completamente conectado a todos os outros da camada anterior.

1.2.3 Aprendizagem Profunda

Na aprendizagem profunda são utilizadas redes neurais profundas para a realização da aprendizagem. Numa rede neural profunda [4], como descrito na secção anterior, os nós estão organizados em múltiplas camadas sucessivas. Cada uma aprende a transformar o input numa representação mais abstrata. A uma rede profunda, por norma, estão associados melhores resultados. No entanto, uma rede demasiado profunda pode acarretar certos problemas que devem ser considerados quando o modelo é construído.

Modelos muito profundos com muitos parâmetros requerem grandes quantidades de dados. Caso não existam dados suficientes, o modelo, apesar de poderoso, irá dar overfit dos dados de treino, resultando num erro de treino bastante baixo, mas, muito provavelmente não se sairá bem com os dados de teste.

Outro problema que pode surgir quando as redes são muito profundas é o vanishing gradiente. Este problema surge no treino de redes neurais artificiais que utilizam métodos

de aprendizagem baseados em gradientes e retropropagação. Nestes métodos [33], cada peso da rede neural recebe uma atualização da função de perda durante o processo de treino. No entanto, quando a rede é demasiado grande, o gradiente aproxima-se rapidamente de 0, impedindo que exista atualizações nos valores dos pesos, podendo afetar o treino adicional da rede neural.

O último problema principal centra-se nas computações e no armazenamento pois o modelo torna-se exponencialmente maior com a profundidade, e como resultado, também o conjunto de dados aumenta exponencialmente, o que faz com que o treino se torne bastante desafiante e custoso em termos de computações e armazenamento.

Assim, é importante conseguir criar um modelo que consiga ter camadas profundas sem ir ao encontro de nenhum dos problemas acima mencionados.

1.2.4 Redes Neurais Convolucionais

As redes convolucionais [14], também conhecidas por redes neurais convolucionais ou CNN, são utilizadas em contextos de processamento de dados constituídos por uma topologia tipo rede, como é o caso das imagens que podem ser vistas como uma rede 2D de píxeis. Ademais, este tipo de redes é bastante utilizado em processamento de imagens devido à capacidade de aprender a partir de padrões, características ou texturas em vez de se basear nas variações de cor. Alguns exemplos do uso das CNNs neste contexto são: classificação de imagens ou rotulagem de pixels, onde a primeira é um processo de classificar a imagem toda e a segunda caracteriza-se por atribuir classes diferentes a píxeis diferentes. Numa CNN, existem duas camadas principais utilizadas com o objetivo de aprendizagem: camada convolucional e camada pooling.

Camada convolucional

A camada convolucional tem como objetivo efetuar a extração de features da imagem de input, realizando a maior parte das computações numa CNN. Teoricamente, esta operação é o integral de duas funções em que uma delas foi deslocada e invertida (Equação 1.5).

$$s(t) = \int x(a)w(t-a)da \tag{1.5}$$

A função x é usualmente referida como input, a função w refere-se ao kernel e o output é comumente referido como mapa de features.

O kernel (ou filtro) irá mover-se ao longo da imagem desde o canto superior esquerdo até ao canto inferior direito e, para cada ponto, irá calcular um valor com base no filtro, usando uma operação de convolução. Os filtros têm como objetivo encontrar propriedades nos dados fornecidos com input. Nas imagens de microscopia de fluorescência de bactérias, os filtros são utilizados para detetar parte de células e fundo que depois serão aplicados para agrupar os píxeis que pertencem à mesma célula.

Se assumirmos que x e w da Fórmula 1.5 estão definidas apenas sobre o inteiro t, é possível definir uma operação de convolução discreta (Fórmula 1.6).

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{1.6}$$

Também é possível realizar operações de convolução em mais do que um eixo ao mesmo tempo. No caso de termos uma imagem 2D I como input, pode ser necessário o kernel 2D K (Fórmula 1.7).

$$S(i,j) = (I * K)(i,j) = \sum_{m} \sum_{n} I(m,n)K(i-m, j-n) \tag{1.7}$$

Camada Pooling

A camada convolucional [41] é uma pilha de mapas de features, um para cada filtro. Quanto maior o número de filtros, maior será a dimensionalidade da convolução e, por consequência, uma dimensionalidade alta indica mais parâmetros. Desta forma, a camada de pooling controla o overfiting ao reduzir, progressivamente, o tamanho espacial das representações para diminuir o número de parâmetros e computações.

Numa CNN, é possível controlar o comportamento da camada convolucional ao especificar o tamanho de cada filtro e do número de filtros. Para aumentar a quantidade de nós numa camada convolucional, é possível amentar o número de filtros e para aumentar o número de padrões, aumenta-se o tamanho do filtro.

Existem outros hiperparâmetros que podem ser ajustados, um deles é o stride da convolução que se caracteriza pelo valor utilizado pelo filtro para se mover ao longo da imagem. Um stride de 1 move o filtro 1 pixel horizontalmente e verticalmente. Neste caso, a convolução torna-se igual à largura e profundidade da imagem de input. Um stride igual a 2 cria uma camada convolucional de metade da largura e altura da imagem. Caso o filtro se estenda para fora da imagem, é possível ignorar esses valores desconhecidos ou substituí-los por zeros.

1.2.5 Autoencoders

Um autoencoder é um método de aprendizagem não supervisionada, [14] composto por uma rede neural treinada para tentar copiar o input para o output. Internamente, tem uma camada escondida h que é utilizada para descrever o código usado para representar o input. A rede pode ser vista como estando dividida em duas partes: a função de codificação h = f(x) e a função de descodificação r = g(h).

Na prática, ter-se um autoencoder que aprendeu a copiar o input, não tem qualquer utilidade. Assim, o codificador tem a utilidade de comprimir dados e o descodificador é utilizado para reproduzir a imagem original. Normalmente, são restringidos de forma a permitir que copiem apenas o input aproximadamente e entradas que se assemelham aos dados de treino. Como o modelo é forçado a priorizar quais os aspetos do input que devem ser copiados, geralmente o autoencoder aprende propriedades úteis dos dados.

1.2.5.1 Autoencoders Convolucionais

Um autoencoder convolucional [41] é uma rede neural (caso especial de um modelo de aprendizagem não supervisionada) que é treinada para reproduzir a imagem de input na camada de output. Uma imagem é enviada para o codificador, que é uma rede convolucional, que produz uma representação de baixa dimensão dessa imagem. O descodificador, que é outro exemplo de uma rede convolucional, recebe a imagem comprimida que saiu do codificador e, a partir dessa imagem, reconstrói a original.

Desta forma, o codificador tem a utilidade de comprimir dados e o descodificador é utilizado para reproduzir a imagem original. Adicionalmente, os autoencoders têm diversas aplicações: compressão de dados, redução de ruido em imagens (produzindo uma imagem mais limpa a partir de uma imagem corrompida), redução de dimensões, procura em imagens, processamento de imagens.

1.2.6 Segmentação de Imagens

Segmentação de imagens [47] é o processo de atribuir uma classe (como pessoa, carro ou árvore) a cada pixel de uma imagem. Este processo pode ser visto como um processo de classificação ao nível do pixel, onde cada pixel é classificado separadamente. Com base nessa propriedade, a segmentação pode ser classificada em dois grupos distintos: **segmentação semântica** onde é atribuída uma classe a cada pixel, mas não existe diferenciação entre os objetos pertencentes à mesma classe e **segmentação de instância** em que é atribuída uma classe a cada pixel e existe diferenciação entre os objetos iguais.

Uma forma mais fácil de segmentação [10] é atribuir píxeis ao mesmo segmento se tiverem a mesma cor. Isto pode ser suficiente nalguns casos, como, por exemplo, em imagens de satélite para medir a quantidade total de área florestal numa determinada área. No entanto, no caso da segmentação de imagens de microscopia de bactérias, esta técnica não é suficiente, pois se tivermos, por exemplo, duas bactérias distintas juntas (ou seja, as bactérias estão em contacto uma com a outra), este processo baseado na cor dos píxeis individuais iria segmentá-las como sendo apenas uma bactéria. Uma vez que um dos principais objetivos deste trabalho é evitar o surgimento deste problema, a segmentação foi feita através dos padrões formados por grupos de píxeis.

1.3 Objetivos e Contribuições

A arquitetura W-Net, baseada na U-Net, proposta por Xide Xia e Brian Kulis em [50] junta duas redes totalmente convolucionais num único autoencoder. A primeira rede tem como objetivo codificar a imagem de input utilizando camadas totalmente convolucionais, numa segmentação k-way. A segunda rede reverte este processo onde, a partir da camada segmentada volta a reconstruir a imagem original. Durante este procedimento, é minimizado quer a função de perda de reconstrução do autoencoder quer a função de perda de corte normalizado suave da rede codificadora. Depois desse processo são efetuados dois

passos de pós-processamento para melhorar a segmentação obtida no autoencoder. O primeiro passo consiste em aplicar um campo aleatório condicional totalmente conectado e, de forma a obter a segmentação final, é aplicado o método de junção hierárquica.

Técnicas recentes de visão computacional têm algumas limitações quando utilizadas na segmentação de imagens de microscopia de fluorescência de bactérias, nomeadamente no que toca a grupos de células juntas ou sobrepostas. Nestes casos, o algoritmo não consegue distinguir com exatidão as diferentes células o que o leva a segmentar duas ou mais células como se fossem apenas uma. A aprendizagem supervisionada profunda tem o potencial de corrigir o problema acima mencionado. No entanto requer grandes quantidades de dados rotulados. O que no caso desta dissertação requer selecionar cada bactéria presente em cada imagem de treino o que é bastante demorado e trabalhoso. Assim, a utilização de métodos não supervisionados têm o potencial de resolver ambos os problemas.

Deste modo, a arquitetura W-Net descrita no início desta secção foi adaptada de forma a acomodar uma variação no número de filtros e no número de camadas utilizadas. A arquitetura W-Net, minimiza a função de perda de reconstrução do autoencoder e a função de perda de corte normalizado suave na camada codificadora. Adicionalmente, no artigo original foram utilizados os seguintes passos de pós processamento para melhorar a segmentação obtida: campo aleatório condicional totalmente conectado e junção hierárquica. Deste modo, para além das alterações mencionadas previamente, esses dois passos de pós-processamento foram também alterados. Posto isto, o primeiro passo passou por combinar as classes do output da rede codificadora para encontrar as melhores máscaras de segmentação. De seguida, a essas máscaras de segmentação foi aplicado o algoritmo de segmentação Watershed para obtenção da máscara de segmentação final. Todas estas alterações tiveram o objetivo de otimizar a rede W-Net e testar se é possível utilizá-la para segmentação de imagens de microscopia de bactérias de forma a resolver ambos os problemas mencionados anteriormente: os erros ocorridos durante o processo de segmentação de grupos de bactérias juntas ou sobrepostas e a necessidade de grandes quantidades de dados manualmente rotulados.

Para treinar o modelo, teve-se acesso a três grupos distintos de imagens de microscopia de bactérias: Phase, Vermelho Texas e DAPI (4',6-diamidino-2-phenylindole). Estas imagens foram criadas a partir dos TIFFs (Tagged Image File Format – formato de arquivo para imagens digitais) originais. O conjunto Phase (Figura 1.2) consiste num conjunto de fotografias com contraste de fase que marca a silhueta das células. Este tipo de contraste é uma técnica utilizada para obter o contraste numa amostra translucida sem manchar a mesma.

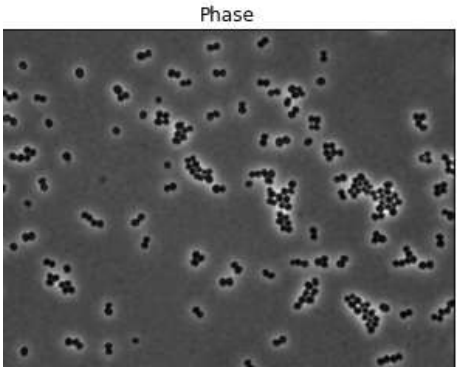


Figura 1.2: Exemplo Conjunto Phase

O Vermelho Texas (Figura 1.3) consiste num grupo de imagens onde se aplicou um corante fluorescente vermelho que é utilizado na coloração de células. Neste caso, este corante está ligado a proteínas da membrana, dando-lhes mais brilho.

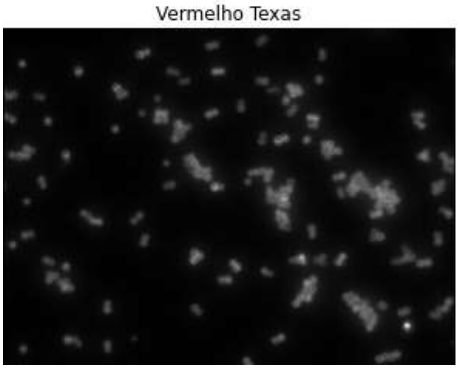


Figura 1.3: Exemplo Conjunto Vermelho Texas

Por último, o DAPI (Figura 1.4) é um marcador fluorescente com a capacidade de se ligar ao ADN. Ao conectar-se às regiões adenina-timina do ADN consegue emitir fluorescência, o que permite que parte do cromossoma das bactérias fique visível.

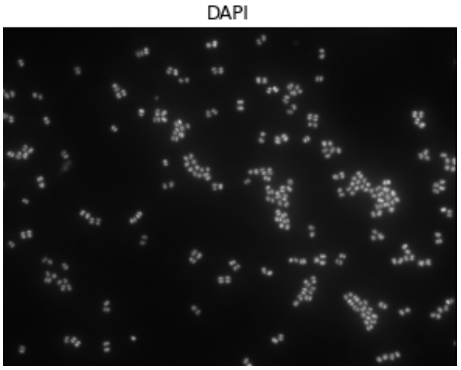


Figura 1.4: Exemplo Conjunto DAPI

O conjunto de dados utilizado para treinar o modelo consistiu no empilhamento dos três grupos de imagens de bactérias anteriormente mencionados. A Figura 1.5 é um exemplo do resultado do empilhamento destes três grupos.

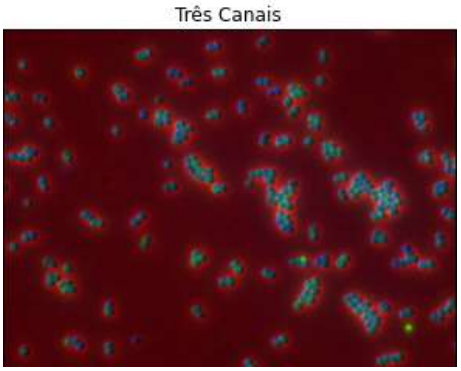


Figura 1.5: Exemplo Três Conjuntos Empilhados

A principal contribuição deste trabalho foi a construção de um modelo de segmentação de imagens de microscopia de fluorescência de bactérias que tentou resolver não só o problema da necessidade de grandes quantidades de imagens manualmente segmentadas, mas também o problema da segmentação de grupos de bactérias que se encontrem próximas ou sobrepostas.

1.4 Organização do documento

Este documento encontra-se dividido em cinco capítulos principais: no primeiro capitulo são descritas as motivações que levaram a este trabalho, algum contexto necessário para o perceber e as contribuições que este trabalho forneceu; no segundo capítulo é feita uma análise à literatura dos algoritmos utilizados na segmentação de imagens, bem como de algumas bibliotecas do Python; no terceiro capítulo é apresentado o trabalho experimental realizado nesta dissertação; o quarto capítulo refere-se à apresentação dos resultados obtidos bem como a sua discussão e, por último, no quinto capítulo são apresentadas as conclusões bem como trabalho futuro a ser realizado.

2

Estado da Arte

As bactérias são seres constituídos por apenas uma célula e, consequentemente, o processo de segmentação de bactérias e de células deverá ser semelhante, o que é particularmente importante [8] para o estudo morfológico das mesmas. A segmentação é utilizada para identificar diversas informações como a forma, estrutura ou o tamanho das mesmas. A segmentação manual de células em imagens de microscopia é um processo caro e que requer um grande consumo de tempo. Desta forma, muitos métodos têm vindo a surgir de forma a automatizar esse processo.

Como referido no capítulo anterior, existem dois tipos de segmentação: **semântica** e de **instância**. Na primeira [45], o mesmo rótulo é atribuído a objetos pertencentes à mesma classe. Consequentemente, quando dois objetos iguais (por exemplo, duas pessoas) que estão em contacto, não é visualizada qualquer divisão entre esses objetos, parecendo que foram segmentados como um único objeto. Desta forma, este tipo de segmentação não é a mais apropriada para o trabalho proposto. Contudo não irá ser totalmente descartada pois existe a possibilidade de se conseguir segmentar corretamente os grupos de células juntas se, por exemplo, for utilizado com algum tipo de pós-processamento. No segundo tipo de segmentação, cada instância de cada objeto recebe um rótulo diferente, garantido que há diferenciação entre dois quaisquer objetos independentemente da sua classe ou posição na imagem. Esta característica é particularmente importante para resolver o problema da segmentação de grupos de células juntas pois, à partida, não requer pós-processamento para o corrigir.

Desta forma, neste capítulo serão apresentados os principais métodos propostos de segmentação de imagens, com particular ênfase nos algoritmos não supervisionados e algoritmos que aplicam segmentação de instância.

2.1 Redes Neurais Convolucionais

Uma típica rede neural convolucional (CNN) consiste numa camada de input, uma camada de output e uma pilha de camadas funcionais intermédias que transformam um input num output e podem ser [31]: Camadas Convolucionais; Camadas Não Lineares; e

Camadas de Pooling.

Camada Convolucional

Nas camadas convolucionais é utilizado o kernel (ou filtro) dos pesos de forma a extrair as features. Existem diversos tipos de camadas convolucionais que podem ser utilizadas na construção de modelos de aprendizagem profunda. Entre eles encontram-se os seguintes: convolução simples, convolução 1x1, convolução achatada, convoluções espaciais e entre canais, convoluções separáveis em profundidade, convoluções agrupadas e convoluções agrupadas aleatórias.

Convolução Simples

Uma convolução é uma operação linear que se caracteriza pela multiplicação de um conjunto de pesos com o input. Uma vez que esta técnica foi desenvolvida para inputs com duas dimensões, a multiplicação é realizada entre o input que é um array de dados e um array de duas dimensões que contém os pesos que é chamado de filtro ou kernel. Este filtro é sempre menor que os dados de input, o que permite que o mesmo conjunto de pesos sejam multiplicados pelos dados de input múltiplas vezes em pontos diferentes do input. Mais especificamente, o filtro é multiplicado por cada subconjunto do array de dados de input, da esquerda para a direita, de cima para baixo. A Figura 2.1, mostra esta operação visualmente.

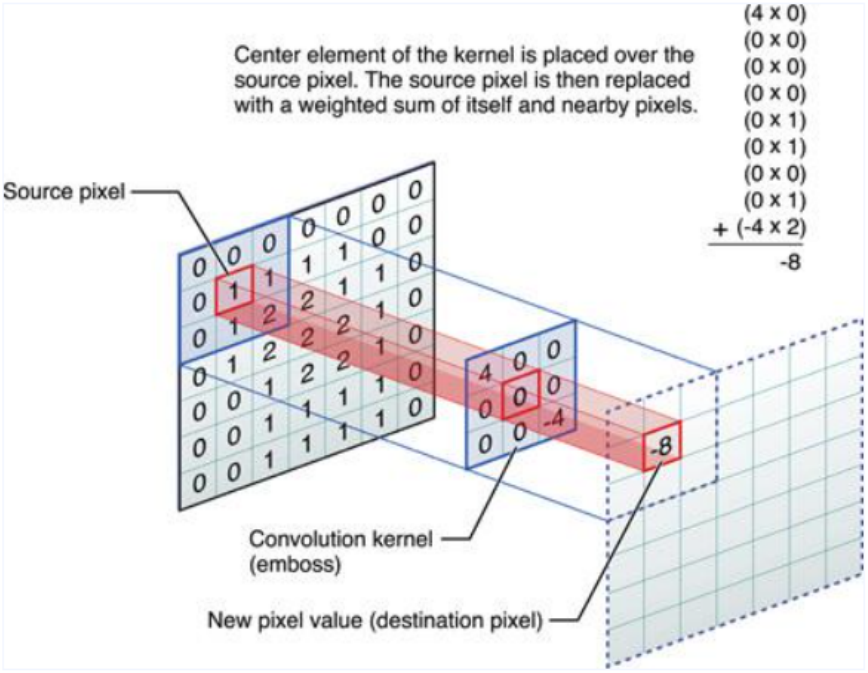


Figura 2.1: Convolução Simples [20]

A Figura 2.2, mostra um outro exemplo [49] em que assumimos que temos uma imagem de tamanho igual a 12x12x3 e que aplicamos uma convolução de 5x5 sem padding

(processo de adicionar zeros à imagem inicial) e de stride (número de passos que movemos em cada passo da convolução) igual a 1 à imagem. Se apenas considerarmos a altura e largura da imagem, o processo de convolução funciona da seguinte forma: são aplicadas multiplicações escalares a cada 25 píxeis, retornando sempre um valor. A imagem resultante tem tamanho igual a 8x8, uma vez que não foi aplicado padding (12-5+1). No entanto, como a imagem é RGB, tendo por isso, três canais, o kernel convolucional também precisa de ter 3 canais. Isto significa que, ao invés de implementar 5x5=25 multiplicações, são realizadas 5x5x3=75 multiplicações de cada vez que o kernel se move. Tal como na interpretação 2D, são realizadas multiplicações de matrizes a cada 25 píxeis, devolvendo um número. Depois de passar pelos kernels de 5x5x3, a imagem 12x12x3 transforma-se numa outra de tamanho 8x8x1.

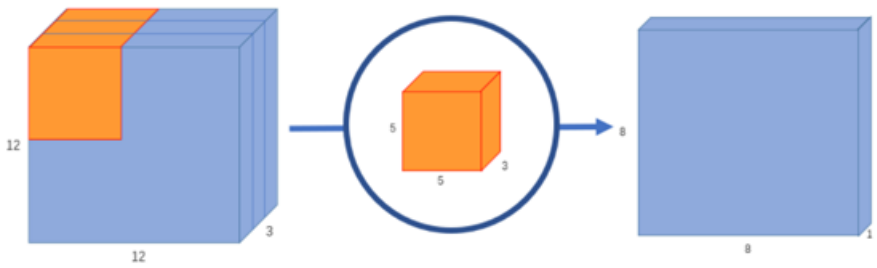


Figura 2.2: Convolução Simples - 3D [49]

No entanto, se desejarmos aumentar o número de canais da imagem de output, basta criar o número de kernels equivalente ao número de canais desejados. Por exemplo, se o objetivo for uma imagem de tamanho 8x8x6, basta utilizarmos 6 kernels para devolverem 6 imagens de tamanho 8x8x1 e depois empilhar essas 6 imagens.

Esta aplicação sistemática do mesmo filtro ao longo da imagem é bastante poderosa no sentido em que se o filtro foi designado para detetar um tipo específico de features do input, então a aplicação sistemática do filtro ao longo de toda a imagem de input permite que o filtro descubra features em qualquer parte da imagem. Esta característica é normalmente referida como invariância de tradução, ou seja, o interesse generalizado de saber se uma dada feature está presente ao invés de saber onde esta se encontra.

Convolução 1x1

A convolução 1x1 foi inicialmente proposta em [25] e muito utilizada por [43]. Este tipo de convolução, ilustrado na Figura 2.3, é caracterizado por aplicar convoluções com filtros de tamanho 1x1 às imagens de input, sendo que as principais características são: aumentar ou reduzir a dimensionalidade e aplicar não-linearidade depois das convoluções. A ideia por detrás deste tipo de convoluções é a seguinte: se tivermos uma imagem de tamanho 32x32x100, onde 100 representa as features, depois de aplicarmos 20 convoluções com filtros de 1x1, obtêm-se imagens com as seguintes dimensões: 32x32x20.

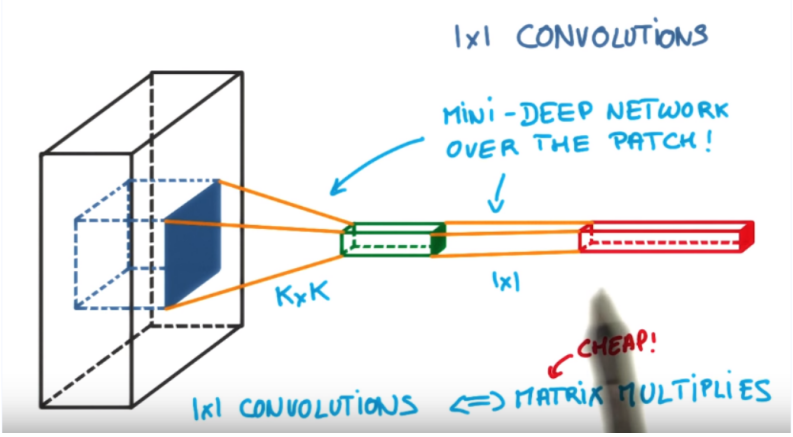


Figura 2.3: Convolução 1x1 [20]

Convolução Achatada

Têm existido diversas propostas para acelerar os diferentes passos da CNN. Em [19] foi proposta uma forma de diminuir a redundância dos filtros nas redes neurais convolucionais. Para tal, na fase de treino os filtros 3D convencionais são separados em três filtros 1D consecutivos: as convoluções são aplicadas ao longo dos canais (laterais), vertical e horizontalmente.

Os pesos numa CNN podem ser descritos como filtros em 4 dimensões: $W \in R^{(CxXxYxF)}$, onde C é o número de canais de input, X e Y as dimensões espaciais do filtro e F o número de filtros ou o número de canais de saída. Uma convolução para cada canal de saída requer um filtro com $W \in R^{(CxXxY)}$ e é descrita pela Fórmula 2.1.

$$F_f(x,y) = I * W_f = \sum_{c=1}^{C} \sum_{x'=1}^{X} \sum_{y'=1}^{Y} I(x, x-x', y-y') W_f(c, x', y') \tag{2.1}$$

Onde f é um index do canal de output, $I \in R^{(CxNxMxF)}$ é o mapa de input, N e M são as dimensões espaciais do input. Uma forma de acelerar a convolução multidimensional é aplicar a separação de um filtro W_f. Para tal, é utilizando o filtro de classificação unitária $\hat{W}_f$ que pode ser separado em produtos cruzados de três filtros unidimensionais como mostrado na Equação 2.2.

$$\hat{W}_f = \alpha_f x \beta_f x \gamma_f \tag{2.2}$$

Neste caso, os vetores convolucionais 1D são filtros laterais α_f: features de convolução em todos os canais; filtros verticais β_f: ao longo da dimensão Y; e horizontais filtros γ_f: ao longo da dimensão X.

No entanto, a separabilidade dos filtros é uma condição forte e a classificação intrínseca do filtro W_f é maior do que na prática. À medida que as dificuldades dos problemas de classificação aumentam, também aumenta o número de componentes principais necessárias à resolução do problema. Os filtros aprendidos em redes profundas distribuíram eigenvalues e aplicam a separação diretamente aos filtros resulta em grandes perdas de

informação. Alternativamente, poderíamos restringir conexões em campos recetivos para que o modelo consiga aprender filtros 1D separados durante o processo de treino. Quando aplicada à Equação 2.1, uma única camada de redes neurais convolucionais é modificada para a Equação 2.3.

$$\hat{F}_f(x,y) = I * \hat{W}_f = \sum_{x'=1}^{X}\left(\sum_{y'=1}^{Y}\left(\sum_{c=1}^{C} I(c,x-x',y-y')\alpha_f(c)\beta_f(x')\right)\gamma_f(y')\right) \tag{2.3}$$

Com esta modificação, o número de parâmetros para calcular cada mapa de features diminui de X*Y*C para X + Y + C, e o número de operações necessárias diminui de M*N*C*X*Y para M*N*(C + X + Y).

Resumidamente, uma convolução achatada, ilustrada na Figura 2.4, são convoluções que são convertidas numa sequência de convoluções 1D.

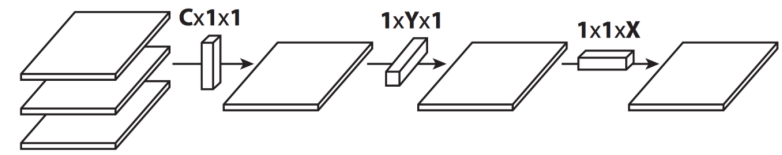


Figura 2.4: Convolução Achatada [20]

Convoluções Espaciais

As convoluções espaciais (Figura 2.5) dividem as operações espaciais numa série de operações independentes. Por outras palavras, significa que se realizam convoluções em dimensões espaciais – largura e altura.

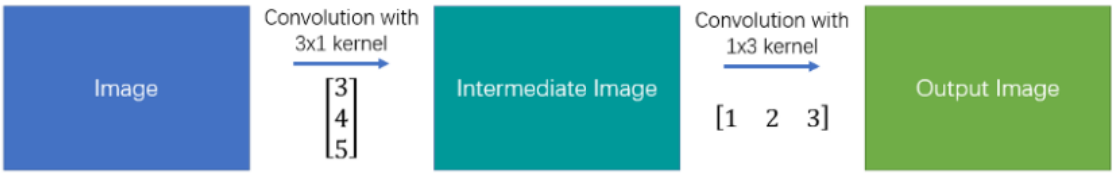


Figura 2.5: Convolução Espacial [49]

Convoluções Separáveis em Profundidade

As convoluções separáveis em profundidade têm em conta não só as dimensões espaciais da imagem de input, mas também as dimensões em profundidade (o número de canais). Uma imagem pode ter três canais (se for RGB) e, depois de se aplicar algumas convoluções, uma imagem pode ter múltiplos canais, sendo que cada canal pode ser visto como sendo uma interpretação particular dessa mesma imagem. Por exemplo, o canal vermelho indica a quantidade de vermelho persente em cada píxel da imagem.

As convoluções separáveis em profundidade [7] realizam uma convolução espacial em profundidade seguida por uma convolução pontual.

Convolução Espacial em Profundidade: atua em cada canal do input de forma independente. Ou seja, a imagem inicial é convolvida sem alterar a profundidade. Por

exemplo, se tivermos uma imagem RGB de 12x12, terá o seguinte tamanho 12x12x3, se aplicarmos três filtros 5x5x1, cada um irá iterar num único canal da imagem, obtendo o produto escalar para cada grupo de 25 píxeis, originando três imagens (uma por cada filtro) de tamanho 8x8x1 (12-5+1). Se empilharmos as imagens resultantes dos três filtros, obtemos uma imagem com tamanho 8x8x3.

Convolução pontual: a convolução original transforma uma imagem 12x12x3 numa imagem 8x8x1 ao aplicar um filtro de 5x5x3 à imagem original. No entanto, se quisermos que a imagem resultante tenha um número diferente de canais, é necessário aplicar um número de kernels igual ao número de canais desejado. Por exemplo, para termos uma imagem 8x8x256, basta aplicar 256 kernels para criar 256 imagens 8x8x1, e de seguida, empilhá-las para criar uma imagem de output com 8x8x256. Neste momento, a convolução em profundidade transformou a imagem de 12x12x3 numa imagem 8x8x3. De seguida, é preciso aumentar o número de canais de cada imagem. A convolução pontual tem este nome porque utiliza um filtro 1x1, ou um filtro que itera sobre cada ponto. Este filtro tem uma profundidade igual ao número de canais da imagem inicial, no caso do exemplo dado igual a 3. Assim, é iterado um filtro 1x1x3 ao longo da imagem 8x8x3, de forma a obter uma imagem 8x8x1. É possível criar 256 filtros 1x1x3 cujo output de cada um é uma imagem 8x8x1. Por último, para obter a imagem final com forma 8x8x256, basta empilhar as imagens resultantes de aplicar os 256 kernel à imagem inicial.

A arquitetura W-Net utiliza este tipo de convoluções (Figura 2.6), sendo que a ideia em aplicar convoluções separáveis consiste [50] em examinar correlações espaciais e correlações de canal cruzado de forma independente – uma convolução em profundidade realiza convoluções espaciais independente em cada canal e, em seguida, uma convolução pontual projeta os canais de features pela convolução em profundidade num novo espaço de canal. Como consequência, a rede ganha desempenho de forma mais eficiente com o mesmo número de parâmetros.

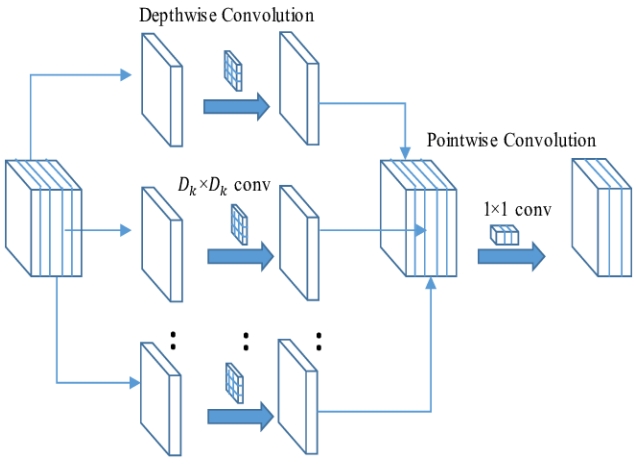


Figura 2.6: Convolução Separável em Profundidade [16]

Convoluções Agrupadas

Introduzida em 2012 no artigo da AlexNet ([22]), a convolução agrupada foca-se em utilizar diferentes conjuntos de filtros de convolução na mesma imagem de forma paralela, utilizando, para isso diferentes GPUs (Figura 2.7). Esta ideia surgiu para contornar problemas associados a GPUs mais fracos e com menos memória.

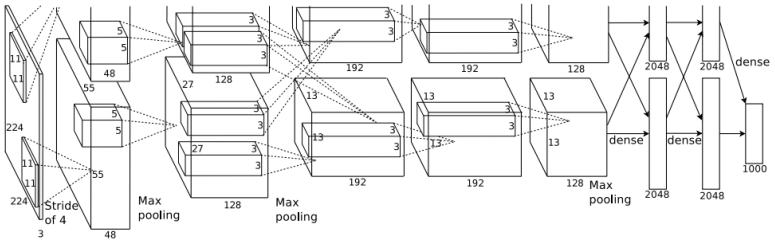


Figura 2.7: Convolução Agrupada [51]

Usando a Figura 2.8 como exemplo, se utilizarmos uma camada convolucional com dois grupos de filtros, pode-se observar que cada um dos filtros na camada agrupada utiliza metade da profundidade, ou seja, utilizando metade dos parâmetros e realizando metade dos cálculos.

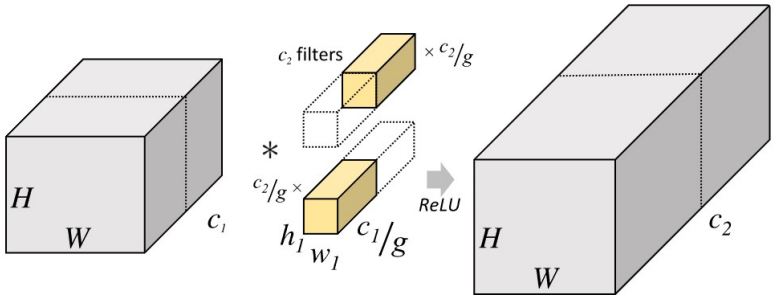


Figura 2.8: Convolução Agrupada Exemplo [18]

Convoluções Agrupadas Aleatórias

As redes neurais convolucionais [52] normalmente consistem em blocos de construção repetidos com a mesma estrutura. Entre eles, as redes do estado da arte como Xception e ResNeXt introduziram convoluções separáveis em profundidade ou convoluções de grupo nos blocos de construção para encontrar um equilíbrio entre a capacidade de representação e custo computacional. No entanto, em [52] repararam que ambos os designs não têm em consideração as convoluções 1x1, que possuem complexidade considerável. Para resolver esse problema, uma solução simples consiste em aplicar conexões esparsas de canal, por exemplo, convoluções de grupo, também em camadas 1x1. Ao garantir que cada convolução opera apenas no grupo de canais de entrada correspondente, a convolução de grupo reduz significativamente o custo de computação. No entanto, se empilharmos várias convoluções agrupadas, existe um efeito colateral: os outputs de um determinado

canal são apenas derivados através de uma pequena fração dos canais de input: a Figura 2.9 (a) ilustra uma situação de duas camadas convolucionais agrupadas empilhadas, o que faz com que os outputs de um certo grupo apenas se relacionam com os inputs desse mesmo grupo. Esta propriedade bloqueia o fluxo de informação entre os diferentes grupos de canais e a representação de conjuntos fracos. Se for permitido que a convolução de grupo obtenha dados de input de grupos diferentes (como mostrado na Figura 2.9 (b)), os canais de input e de output serão totalmente relacionados. Mais concretamente, para o mapa de features gerado na camada de grupo anterior, é possível dividir os canais em cada grupo em vários subgrupos e, depois, alimentar cada grupo na camada seguinte com diferentes subgrupos. Isto pode ser implementado com eficiência utilizando uma operação de shuffle de canais (Figura 2.9 (c)): se tivermos uma camada convolucional com g grupos em que a saída tem g x n canais, o que é feito é o seguinte: primeiro a dimensão do canal de output é alterada para (g, n), transpondo e, de seguida, é achatado de volta como a entrada da próxima camada. Esta operação continua a ter efeito mesmo em casos em que ambas as convoluções tenham números diferentes de grupos. Para além disso, o shuffle de canais é diferenciável, o que significa que pode ser utilizado em redes para treino de ponta a ponta. A operação de mistura de canais torna possível construir estruturas mais poderosas com várias camadas convolucionais de grupo.

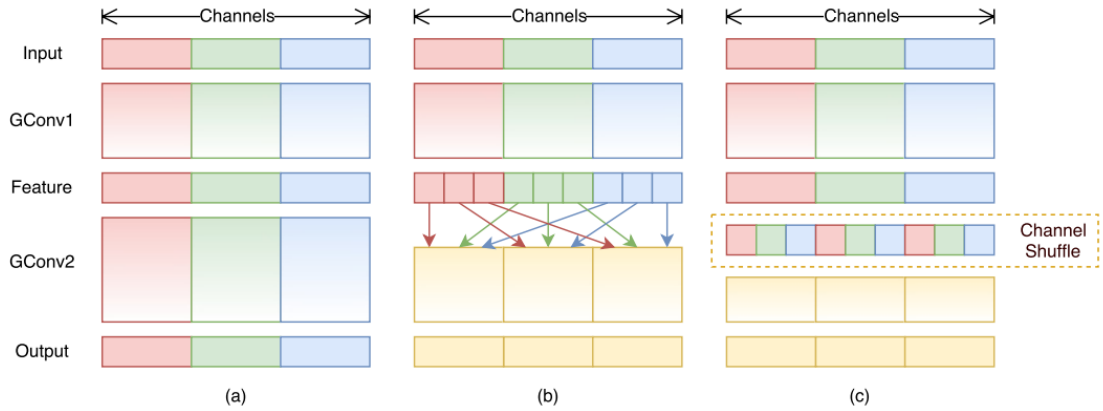


Figura 2.9: Convolução Agrupada Aleatória [52]

Camadas não lineares

As camadas não lineares aplicam uma função de ativação aos mapas de features de forma a introduzir não linearidade na rede.

Camadas pooling

As camadas de pooling substituem uma pequena vizinhança de um mapa de features com algumas informações estatísticas (média, máximo, etc) sobre a vizinhança e reduzem a resolução espacial.

Os nós nas camadas são conectados localmente, ou seja, cada nó recebe inputs ponderados de uma pequena vizinhança, conhecida como campo recetivo, de nós da camada

anterior. Ao empilhar camadas, as mais superiores aprendem features de campos recetivos cada vez mais amplos. A principal vantagem computacional das CNNs é que todos os campos recetivos numa camada compartilham pesos, resultando num número significativamente mais pequeno de parâmetros do que as redes totalmente conectadas.

Outra vantagem das CNNs é a capacidade de detetar as features principais sem qualquer supervisão humana. Para além disso, é computacionalmente mais eficiente que os métodos propostos antes das CNNs pois utiliza convoluções e operações de Pooling e executa a partilha de parâmetros. Isto permite que os modelos CNN funcionem em qualquer dispositivo, tornando-os universalmente atraentes. Para além disso, as redes neurais convolucionais conseguem aprender a partir de padrões, características ou texturas em vez de se basearem nas variações de cor, como muitos dos outros métodos.

No entanto, as CNNs também têm algumas desvantagens [53] onde a principal centra-se na ideia destas não conseguirem atingir precisão elevada na segmentação de imagens pois filtram as informações de baixo nível. Isto faz com que os resultados da segmentação sejam, geralmente, limites não nítidos e formas semelhantes a machas para filtros convolucionais. Este problema promoveu o desenvolvimento de CNNs para a visão computacional de baixo nível, onde combinar as features das camadas superior e inferior é a chave para resolver o problema mencionado.

As redes neurais convolucionais revolucionaram as tarefas de processamento de imagens como a segmentação de imagens ao fornecerem a base para a criação de diversas redes e arquiteturas para esse fim, onde se destacam as seguintes: Mask-RCNN, Autoencoders Convolucionais, SegNet, U-Net, e W-Net. Estas arquiteturas serão explicadas com mais detalhe nas próximas secções.

2.1.1 Mask R-CNN

O R-CNN [13] é um método para deteção de objetos composto por três módulos: 1) Gera propostas de regiões independentes de categorias que definem o conjunto de regiões candidatas disponíveis para o detetor; 2) Rede neural convolucional que extrai um vetor de features de comprimento fixo para cada região; e 3) Conjunto de SVMs lineares específicos da classe. Apesar dos bons resultados, este algoritmo tem diversas desvantagens [12]: 1) O treino é um pipeline com múltiplos estágios; 2) O treino é caro no espaço e no tempo; e 3) A deteção de objetos é lenta.

O método Fast R-CNN foi proposto com o objetivo de eliminar as desvantagens mencionadas anteriormente. Neste, uma imagem e múltiplas regiões de interesse (RoIs) são o input de uma rede totalmente convolucional. Cada RoI é agrupado num mapa de features de tamanho fixo e, em seguida, mapeado para um vetor de features por camadas totalmente conectadas.

O método anterior já mostrava bons resultados em termos de tempo e de deteção de objetos. Não obstante, foi proposto um algoritmo ainda mais eficiente chamado Faster R-CNN que estende o método anteriormente referido. Este [37] é um sistema de deteção

de objetos composto por dois módulos. O primeiro é uma rede profunda totalmente convolucional que propõe regiões de interesse e o segundo é um detetor Fast R-CNN que utiliza as regiões propostas.

O Mask R-CNN [15] é um método para segmentação de instância de objetos baseado no algoritmo de deteção de objetos Faster R-CNN. Esta abordagem deteta de forma eficiente objetos numa imagem enquanto, simultaneamente, gera uma máscara de segmentação de alta qualidade para cada instância. Este método estende o Faster R-CNN ao adicionar um ramo que prevê uma máscara para o objeto. Desta forma, o Mask R-CNN adota o mesmo procedimento de duas fases do Faster R-CNN, cuja primeira fase é idêntica em ambos os métodos. Já na segunda fase, em paralelo à previsão da classe e deslocamento da caixa, também é produzida uma máscara binária para cada RoI.

Este método tem vindo a ser utilizado com muita frequência na segmentação de imagens pois demonstrou ótimos resultados na segmentação de instância e deteção de objetos. No entanto, é um método consideravelmente lento, sendo difícil de ser utilizado em diversas aplicações.

2.1.2 Autoencoders

Um autoencoder [48] é um método de aprendizagem não supervisionada utilizado com o propósito de extração de features e redução da dimensionalidade dos dados. Este [11] é composto por um codificador que codifica as representações do input para uma codificação intermédia de dimensão inferior e um descodificador que tenta reconstruir a entrada original a partir da representação intermédia. O codificador [48] recebe um input x com dimensão d, e mapeia-o para uma representação escondida y, de dimensão r, utilizando uma função de mapeamento determinístico (Equação 2.4).

$$y = f(Wx + b) \tag{2.4}$$

Onde os parâmetros W e b têm de ser aprendidos pelo codificador e representam os pesos e bias, respetivamente, associados à camada que recebe o input x.

O descodificador recebe o output y do codificador e utiliza a mesma função de mapeamento f de forma a fornecer uma reconstrução z que tem de ser da mesma forma de x. A partir da Fórmula 2.4, o output do descodificador é dado pela Equação 2.5.

$$z = f(W'y + b') \tag{2.5}$$

Onde os parâmetros W' e b' são, respetivamente, os pesos e bias associados à camada do descodificador. No final, a rede tem de aprender os parâmetros W, W', b e b' de forma que z seja o mais parecido possível a x.

Resumidamente, os autoencoders [29] são construídos para reduzir a dimensão dos dados ao projetar as representações de entrada num espaço de dimensão mais pequeno que o do input. Como tal, num autoencoder é utilizado o mesmo número de nós nas camadas de input e output e um menor número de nós nas camadas escondidas.

Nos últimos anos, inúmeros métodos baseados em autoencoders foram propostos para o processamento de imagens. A principal razão para esta grande aderência encontra-se no facto dos autoencoders serem um método de aprendizagem profunda não supervisionada, permitindo treinar a rede sem quaisquer dados rotulados. Para além disso, a representação latente dos autoencoders permite uma baixa dimensionalidade, capturar a estrutura e compressão de dados.

Existem diversos tipos de autoencoders tais como: autoencoder de redução de ruído ou convolucional. Neste trabalho, vai-se implementar o segundo tipo pois tem vindo a ser utilizado em segmentação de diversos tipos de imagens, onde obteve bons resultados. Para além disso, é treinado com dados sem rótulos e aprende a partir de padrões, características ou texturas. Estas propriedades são extremamente importantes quando os dados de input são imagens de microscopia de bactérias onde a rotulação das mesmas é uma tarefa extremamente trabalhosa.

2.1.2.1 Autoencoders Convolucionais

Um autoencoder simples caracteriza-se por uma rede de feed-forward com três camadas onde as conexões entre as camadas estão totalmente conectadas. No entanto [54], este tipo de autoencoder ignora a estrutura 2D da imagem introduzindo redundância nos parâmetros e fazendo com que as features sejam globais.

O autoencoder convolucional estende a estrutura básica do autoencoder simples ao alterar as camadas totalmente conectadas por camadas convolucionais, preservando a localização espacial por meio da partilha dos pesos entre todos os locais do input.

Arquitetura Base

Um autoencoder convolucional [23] é constituído por camadas convolucionais que aplicam operações de convolução em cada ponto da imagem, passando o resultado para a próxima camada. Nestas camadas são aplicados filtros de forma a extrair as features das imagens.

Cada camada convolucional é seguida por uma função de ativação cujo objetivo é introduzir não linearidade ao sistema pois, durante as camadas de convolução apenas se realizam operações lineares. Após a função de ativação, é aplicada uma camada de Max Pooling com o objetivo de controlar o overfiting e reduzir o número de pesos, o que reduz os custos computacionais.

No processo de reconstrução são utilizadas **camadas de up-sampling**, caracterizadas por não terem pesos e duplicarem a dimensão do input através da repetição das colunas e linhas dos dados; e **camadas convolucionais** que são seguidas por uma função de ativação sigmoid.

Existem diversas variantes deste autoencoder para realizar a segmentação de imagens. As diferenças principais entre essas variantes encontram-se no número de camadas utilizadas, no tamanho do filtro e nas funções de perda utilizadas.

Num autoencoder convolucional, como em qualquer outro tipo de autoencoder, podem existir uma (autoencoder simples) ou mais (autoencoder profundo) camadas escondidas. Apesar dos autoencoders serem muitas vezes treinados com apenas um codificador e um descodificador, a utilização de um autoencoder profundo tem diversas vantagens, nomeadamente [14], a profundidade pode reduzir exponencialmente o custo computacional de representar algumas funções bem como a quantidade de dados de treino necessários para aprender algumas funções.

2.1.2.2 Autoencoder Convolucional 2D versus Autoencoder Convolucional 3D

A grande diferença entre o autoencoder convolucional 2D e o 3D encontra-se na rede neural convolucional que constitui ambos os autoencoders. O primeiro é constituído por CNNs 2D que, normalmente, são utilizadas para o processamento de imagens RGB, ou seja, com 3 canais. O segundo é constituído por CNNs 3D que equivalem às 2D mas em 3D: recebem como input um volume 3D ou uma sequência de frames 2D. As redes 3D são um modelo poderoso para aprendizagem de representações de dados volumétricos. Desta forma, neste trabalho irão ser utilizados autoencoders convolucionais 2D.

2.1.2.3 Autoencoder Convolucional Profundo

Nestes autoencoders, o processo de codificação-descodificação [48] é realizado com recurso às redes neurais convolucionais. Ao contrário das redes neurais convencionais, onde é possível definir o tamanho do output desejado, as redes neurais convolucionais são caracterizadas por um processo de Down Sampling, conseguido pelas camadas de Pooling que são incorporadas na sua arquitetura. Este processo de Down Sampling induz perda da informação espacial enquanto a rede se torna mais profunda. Mas existem diversas soluções que combatem esse problema, nomeadamente, a utilização de padding [23] que preserva todas as informações acerca do volume do input original, permitindo extrair as features de baixo nível necessárias para a segmentação; ou substituir as redes neurais de convolução [48] por um autoencoder convolucional profundo. Como num autoencoder comum, este é composto por um codificador que vai conter um processo de Down Sampling e por um descodificador que realiza up-sample da representação até reconstruir o tamanho original. A solução final da rede pode ser descrita pela Equação 2.6.

$$(W, W', b, b') = \arg \min_{W,W',b,b'} L(xz) \tag{2.6}$$

Onde o z representa o output do descodificador que é descrito pela Equação 2.5 e o x representa a imagem original. A rede aprende os parâmetros da Equação 2.6 de forma a minimizar a função de custo.

2.1.2.4 Autoencoder Convolucional Disperso

A principal diferença [24] entre um autoencoder convolucional disperso e o típico autoencoder convolucional encontra-se na função de perda. Neste caso, a função é composta por uma função de perda de construção, um termo de regularização e uma restrição de dispersão vitalícia. A dispersão pode ser definida em termos de **dispersão populacional** que impõe a dispersão em todos os nós escondidos da camada escondida para garantir representações dispersas dos dados de input; e **dispersão do tempo de vida** que controla a frequência de ativação de cada nó escondido de um grupo de dados. Assim, a função de perda é representada pela Equação 2.7.

$$L_{sparse}(W_1, W_2, b_1, b_2, x) = \frac{1}{2}\| y - x \|^2 + \sum_{l=1}^{2}\sum_{i=1}^{n_i}\sum_{j=1}^{n_h}(W_{ij}^{l})^2 + \beta\sum_{j=1}^{n_h} KL(\rho \,\|\, \hat{\rho}) \tag{2.7}$$

Onde o primeiro termo é um erro médio da soma dos quadrados, o segundo é uma regularização que ajuda a evitar o sobre ajuste e o terceiro representa uma dispersão vitalícia que está na forma de divergência Kullback-Leibler (KL), que é calculado através da Equação 2.8.

$$KL(\rho \,\|\, \hat{\rho}) = \rho \log\left(\frac{\rho}{\hat{\rho}}\right) + (1-\rho)\log\left(\frac{1-\rho}{1-\hat{\rho}}\right) \tag{2.8}$$

Onde ρ é uma constante da restrição de dispersão e $\hat{\rho}$ é a ativação média da camada escondida. A divergência KL é utilizada para medir a diferença de distribuição entre ρ e $\hat{\rho}$ que tende a ser a mesma quando a divergência KL está próxima de zero.

Ao envolver esta divergência na função de perda e tornar ρ num valor pequeno, a ativação média de cada nó escondido é limitada e a sua dispersão de vida pode ser garantida.

No caso do autoencoder convolucional disperso, o processo de treino alterna entre cálculos progressivos e retropropagação, iterativamente, até encontrar as iterações máximas predefinidas ou o erro ser menor que os valores definidos. As features são extraídas de cada píxel da imagem por meio de um autoencoder convolucional disperso e serão utilizadas posteriormente para agrupamento e fusão de regiões.

2.1.3 SegNet

O VGG16 é um modelo baseado nas redes neurais convolucionais projetado para classificação de objetos. O SegNet [3] foi desenhado para realizar segmentação semântica ao nível do píxel de forma bastante eficiente, baseando-se na arquitetura VGG16. A arquitetura do SegNet, que se encontra ilustrada na Figura 2.10, caracteriza-se por uma rede de codificação e a correspondente rede de descodificação, seguida por uma camada de classificação ao nível do píxel.

A rede de codificação consiste em 13 camadas convolucionais que correspondem às 13 primeiras camadas convolucionais na rede VGG16. A rede codificadora realiza

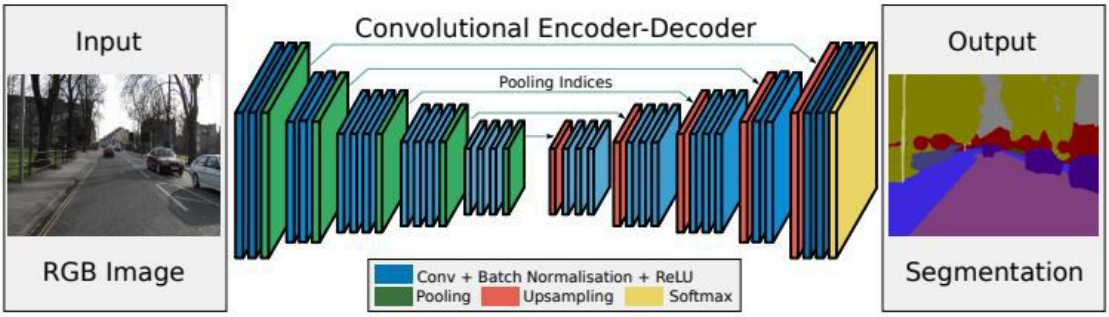


Figura 2.10: Arquitetura SegNet [3]

convolução com um banco de filtros para produzir um conjunto de mapas de features aos quais lhes são aplicados, de seguida, batch normalization e não linearidade através do ReLU. Depois, é aplicado Max Pooling com uma janela de 2x2 e stride 2 e o output resultante é sub-sampled por um fator de 2.

Apesar de várias camadas de Max Pooling e subsampling permitirem alcançar uma maior variação de translação para classificação robusta, há uma perda de resolução dos mapas de features. A representação da imagem com maior número de perdas (detalhes de limite) não é benéfica para a segmentação em que o delineamento do limite é importante. Portanto, é necessário capturar e armazenar informações do limite dos mapas de features do codificador. Para tal, o método utilizado envolve armazenar apenas os índices do Max Pooling, ou seja, as localizações do valor de características máximas em cada janela de Pooling são memorizadas para cada mapa de features.

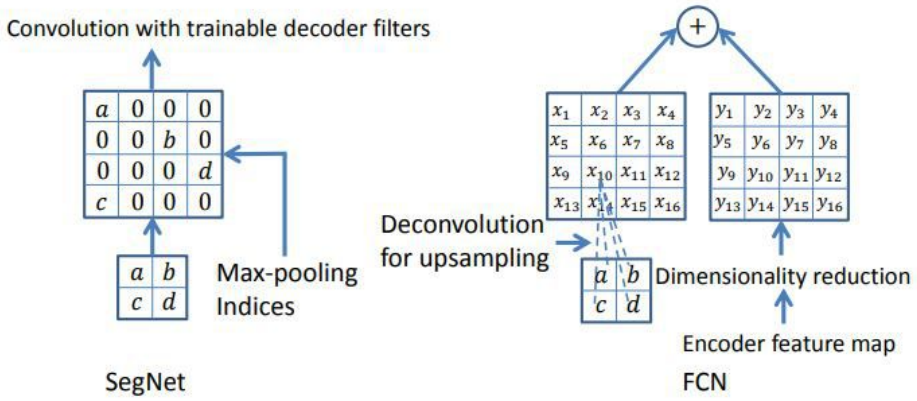


Figura 2.11: Método de Memorização Índices Max-Pooling [3]

A rede de decodificação é a componente chave do SegNet que realiza Up Sampling do mapa de features que recebe como input utilizando, para isso, os índices do Max Pooling memorizados do mapa de features do codificador correspondente. Reutilizar esses índices no processo de descodificação tem várias vantagens práticas: 1) Melhora a delimitação dos limites; 2) Reduz o número de parâmetros ativando o treino de ponta-a-ponta; e 3) Pode ser incorporada em qualquer arquitetura codificador-descodificador. Esta etapa produz

um mapa de features disperso e a técnica de descodificação é ilustrada na Figura 2.11. Estes mapas de features são depois convolvidos com filtros de forma a produzir mapas de features densos e, de seguida, é aplicado batch normalization a cada um desses mapas de features. A representação da feature com maior dimensionalidade é treinada e enviada para um classificador de softmax, que classifica cada píxel de forma independente. O output do classificador softmax é uma imagem do canal K de probabilidades, onde K é o número de classes. A segmentação prevista corresponde à classe com probabilidade máxima em cada píxel.

Apesar desta arquitetura realizar segmentação semântica ao nível do píxel, poderá ser útil neste trabalho. Existe uma possibilidade de, ao realizar segmentação semântica de pequenas janelas da imagem e com pós-processamento, a SegNet conseguir identificar facilmente as células presentes na imagem. Desta forma, apesar de não ser a primeira opção, este algoritmo não será totalmente descartado, havendo a possibilidade de ainda vir a ser utilizado.

2.1.4 U-Net

As redes totalmente convolucionais [39] caracterizam-se por transformar os píxeis das imagens em categorias de píxeis. Esta arquitetura foi modificada e estendida de forma a funcionar com um pequeno número de imagens de treino e levando a segmentação mais precisa.

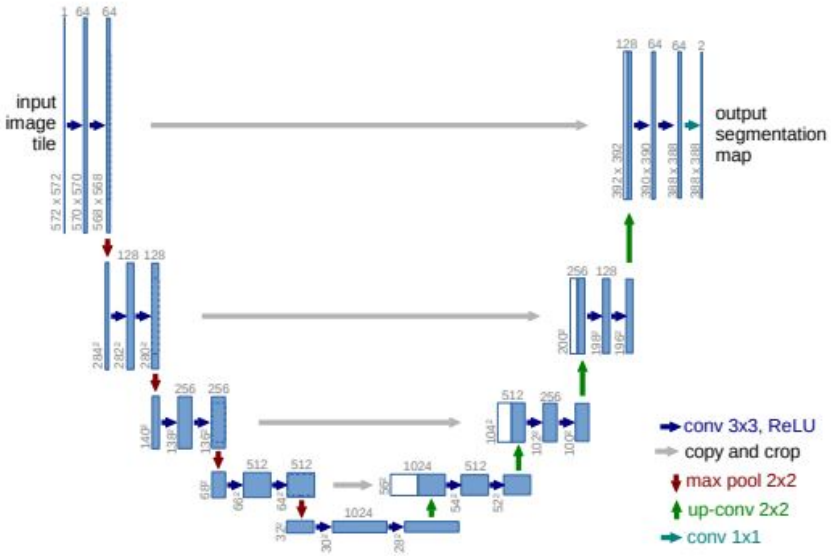


Figura 2.12: Arquitetura U-Net [39]

A arquitetura da rede U-Net está ilustrada na Figura 2.12 e consiste num caminho de contração (lado esquerdo) e num caminho expansivo (lado direito). O caminho de contração segue a típica arquitetura de uma rede convolucional que consiste em aplicar,

repetidamente, duas convoluções 3x3, cada uma seguida por ReLU e uma operação de Max Pooling 2x2 com stride de 2 para o Down Sampling. Cada passo no caminho expansivo consiste em aplicar Up Sampling ao mapa de features seguido por uma convolução 2x2 que divide ao meio o número de canais de features. De forma a evitar a perda dos píxeis das bordas, é necessário aplicar cropping aos mapas de features provenientes do caminho de contração.

Esta arquitetura, apesar de necessitar de poucos dados rotulados, continua a ser um método de segmentação supervisionada, o que vai contra um dos objetivos principais deste trabalho que é a utilização de algoritmos não supervisionados para a realização de segmentação de imagens. No entanto, esta arquitetura teve bastantes bons resultados na segmentação de imagens e, por isso, esteve na base de muitas outras arquiteturas que foram surgindo desde então, como é o caso da W-Net que é explicada de seguida.

2.1.5 W-Net

A rede U-Net [50] é composta por dois caminhos: o caminho de contração onde o contexto é capturado e o caminho de descontração onde ocorre localização precisa. A arquitetura W-Net utiliza ideias recentes de métodos supervisionados de segmentação semântica para a realização de segmentação de imagens de forma não supervisionada. Mais especificamente, a W-Net modificou a arquitetura U-Net de forma a reconstruir as imagens originais do input ao mesmo tempo que prevê uma máscara de segmentação sem qualquer rotulação nos dados de input. Em particular, concatena duas redes totalmente convolucionais (cada uma semelhante à arquitetura U-Net) num autoencoder para realização de segmentação totalmente não supervisionada de imagens.

2.1.5.1 Arquitetura da Rede

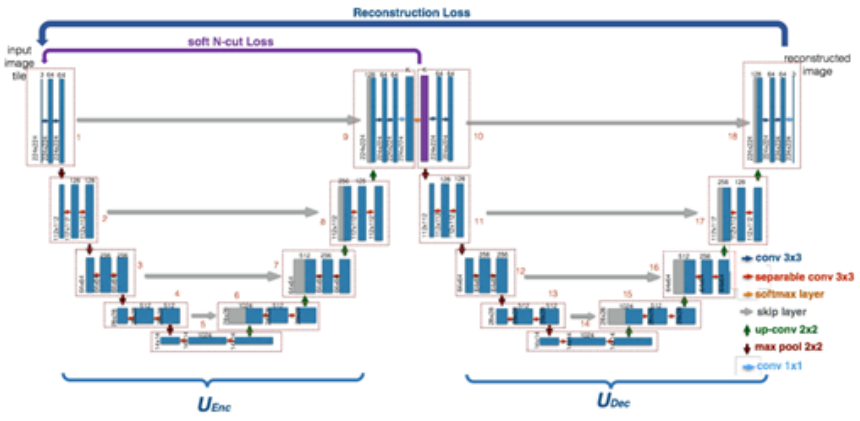


Figura 2.13: Arquitetura W-Net [50]

A arquitetura da W-Net está ilustrada na Figura 2.13 e encontra-se dividida em duas redes distintas: U_{Enc} (parte esquerda) e a correspondente U_{Dec} (lado direito). Como referido anteriormente, a típica arquitetura em forma de U da rede U-Net descrita na secção

anterior foi modificada e estendida para uma arquitetura em forma de W de forma a reconstruir as imagens de input originais ao mesmo tempo que prevê mapas de segmentação sem qualquer informação acerca dos rótulos. A arquitetura W-Net da Figura 2.13 tem 46 camadas convolucionais que estão estruturadas em 18 módulos marcados com retângulos vermelhos. Cada módulo consiste em duas camadas convolucionais 3x3, cada uma seguida por não linearidade ReLU e batch normalization. Os primeiros nove módulos formam a base para a previsão densa da rede e os segundos nove módulos correspondem ao descodificador de reconstrução.

A primeira rede, U_{Enc}, corresponde ao codificador e tem como objetivo mapear a imagem de input numa camada densa de segmentação k-way ao nível do píxel, utilizando camadas totalmente convolucionais. Esta camada terá o mesmo tamanho espacial ao invés de um espaço de dimensão inferior. Para isso, a U_{Enc} é composta por dois caminhos:

- **Caminho de contração** (a primeira metade) para capturar o contexto dos dados fornecidos. Este caminho começa com um módulo inicial que realiza convoluções nas imagens de input que têm o seguinte tamanho: WIDTHxHEIGHTxCHANNEL. Os primeiros dois valores correspondem à largura e altura da imagem, respetivamente, e o último valor diz respeito ao número de canais. Na Figura 2.13, os tamanhos do output são reportados para uma imagem exemplo que é recebida como input que tem uma resolução de 224x224x3. Todos os módulos deste caminho são caracterizados por um conjunto de camadas composto por duas camadas convolucionais, cada uma seguida por ativação ReLU e batch normalization. Cada módulo está ligado ao seguinte através de Max Pooling de 2x2 e o número de canais de features é reduzido para metade em cada passo de Down Sampling;

- **Caminho de expansão** (a segunda metade) ativa a localização precisa, como na arquitetura U-Net original e, para cada módulo deste caminho, é criado um conjunto de camadas composto por: duas camadas convolucionais seguidas, cada uma, por uma camada de ativação ReLU e batch normalization. Cada conjunto é seguido por uma camada de Up Sampling, de forma a duplicar o número de canais de features.

Como no modelo U-Net, o input de cada módulo do caminho contrativo é também utilizado na saída do módulo correspondente no caminho expansivo – skip connections – para recuperação das informações espaciais perdidas devido à redução de resolução após a aplicação de Down Sampling. Mais concretamente, o resultado de cada módulo do caminho contrativo vai ser concatenado ao resultado de cada módulo do caminho expansivo para ser utilizado como input no módulo seguinte.

A camada convolucional final do U_{Enc} é uma convolução de 1x1 seguida por uma camada softmax. Esta convolução mapeia cada vetor de features para o número desejado de classes K, e, de seguida, a camada softmax redimensiona-os para que os elementos do output K fiquem compreendidos entre 0 e 1 e cuja soma seja igual a 1.

A segunda rede, U_{Dec}, funciona como um descodificador, revertendo o processo realizado pelo U_{Enc}, partindo da camada de segmentação prevista, reconstrói a imagem original. A arquitetura do U_{Dec} é bastante semelhante à do U_{Enc}, com exceção que utiliza o output do U_{Enc} cujo tamanho é igual a WIDTHxHEIGHTxK (na Figura 2.13, o tamanho da imagem de input exemplo seria equivalente a 224x224xK). A camada convolucional final do U_{Dec} é uma convolução 1x1 para mapear o vetor de features de volta para a reconstrução do input original, seguida por uma camada sigmoide.

Uma modificação importante na arquitetura apresentada em [50] é que todos os módulos, com exceção dos módulos 1, 9, 10 e 18, utilizam camadas de convolução separáveis em profundidade. Uma operação de convolução separável em profundidade consiste numa convolução em profundidade e noutra pontual. A ideia desta operação é examinar correlações espaciais e correlações de canal cruzado independentemente – uma convolução em profundidade realiza convoluções espaciais independentes em cada canal e, em seguida, uma convolução em profundidade que projeta os canais de features pela convolução em profundidade no novo espaço de canal. Isto permite à rede ficar com um desempenho mais eficiente com o mesmo número de parâmetros. Na Figura 2.13, as setas azuis representam camadas de convolução e as setas vermelhas indicam convoluções separáveis em profundidade. O facto de a rede não incluir nenhuma camada totalmente conectada, permite-lhe aprender imagens arbitrariamente grandes e construir uma previsão de segmentação do tamanho correspondente.

Uma das formas de avaliar se um modelo está a construir previsões corretas é através da medição do seu desempenho ao verificar quão precisas são as decisões do modelo, o que exige encontrar uma forma de medir a diferença entre as previsões e os valores reais correspondentes. As funções de perda têm como objetivo medir a distância entre o valor real e o valor estimado, mapeando as decisões para os seus custos associados. No caso da W-Net descrita em [50], foi minimizado, ao mesmo tempo, a função de perda de reconstrução do autoencoder bem como a função de perda de corte normalizado suave na camada de codificação. De forma a atingirem os mesmos resultados que a literatura neste campo, Xide Xia and B. Kulis [50] aplicaram dois passos de pós-processamento: o primeiro aplica um campo aleatório condicional totalmente conectado que suaviza os segmentos do output do autoencoder e o segundo aplica um método de junção hierárquica para obter a segmentação final. Estes dois passos de pós-processamento serão descritos com mais detalhe no subcapítulo 2.1.5.4.

2.1.5.2 Perda de Corte Normalizada Suave

O corte normalizado [42] caracteriza-se pela aplicação do argmax ao output do U_{Enc} para a obtenção de uma previsão da classe K, sendo que cada output do codificador é uma previsão densa de WIDTHxHEIGHTxK normalizada. Em [50], é utilizado corte normalizado ($Ncut$) como critério global para segmentação, calculado por 2.9.

$$Ncut_V = \sum_{k=1}^{K} \frac{cut(A_k, V - A_k)}{assoc(A_k, V)} = \sum_{k=1}^{K} \frac{\sum_{u \in A_k, v \in V - A_k} w(u,v)}{\sum_{u \in A_k, t \in V} w(u,t)} \tag{2.9}$$

Onde A_k é o conjunto de píxeis do segmento k, V é o conjunto de todos os píxeis e w mede o peso entre dois píxeis.

No entanto, o cálculo do argmax requer saber de forma absoluta quais os pixéis que pertencem a cada classe, o que não é diferenciável e que, por isso, torna impossível calcular o gradiente correspondente durante a retropropagação. Assim sendo, Xide Xia and B. Kulis definiram em [50] uma função de perda baseada na função de perda N_{cut} mais suave e diferenciável que permite atualizar os gradientes durante a retropropagação. Esta nova função denomina-se por corte normalizado suave e caracteriza-se por utilizar a probabilidade de distribuição (softmax) para avaliar se os píxeis pertencem a um dado segmento ou não. É dada pela expressão matemática 2.10.

$$\begin{aligned} J_{soft-Ncut}(V,K) &= \sum_{k=1}^{K} \frac{cut(A_k, V - A_k)}{assoc(A_k, V)} = K - \sum_{k=1}^{K} \frac{assoc(A_k, A_k)}{assoc(A_k, V)} \\ &= K - \sum_{k=1}^{K} \frac{\sum_{u \in V, v \in V} w(u,v) p(u = A_k) p(v = A_k)}{\sum_{u \in A_k, t \in V} w(u,t) p(v = A_k)} \\ &= K - \sum_{k=1}^{K} \frac{\sum_{u \in V} p(u = A_k) \sum_{u \in V} w(u,v) p(v = A_k)}{\sum_{u \in V} p(u = A_k) \sum_{t \in V} w(u,t)} \end{aligned} \tag{2.10}$$

Onde:

- $p(u = A_k)$ mede a probabilidade do nó u pertencer à classe A_k, o que é calculado diretamente pelo codificador;
- K representa o número total de classes;
- $assoc(A_k, A_k)$ representa a associação entre píxeis do mesmo segmento;
- $assoc(A_k, V)$ representa a normalização;
- $w(u,v)$ representa o peso entre os pixéis u e v.

É possível obter, de forma direta, todos os parâmetros acima descritos com exceção dos pesos w(u, v). O cálculo destes pesos baseia-se não só na distância espacial entre os píxeis u e v, mas também na diferença de cor dos mesmos. Ou seja, no caso de píxeis próximos e da mesma cor, dever-se-ia assumir que ambos os píxeis pertencem à mesma região. Por outro lado, caso um destes parâmetros essa suposição poderá não ser verdade. Assim, o cálculo dos pesos [50] foi baseado na Fórmula 2.11.

$$w_{u,v} = e^{\frac{-\|F(u)-F(v)\|_2^2}{\sigma_I^2}} * \begin{cases} e^{\frac{-\|X(u)-X(v)\|_2^2}{\sigma_X^2}} & \text{se } -\|X(u)-X(v)\|_2 < r \\ 0 & \text{caso contrário} \end{cases} \tag{2.11}$$

Onde:

- $F(u) - F(v)$ é a diferença dos vetores de features baseados na informação de intensidade, cor ou textura dos píxeis;

- $X(u) - X(v)$ corresponde à diferença espacial entre os pixéis u e v. Por outras palavras, representa a distância espacial entre cada dois píxeis da imagem;

- r corresponde à distância máxima entre cada dois píxeis que se deve considerar. Ou seja, vamos considerar os pesos para todos os píxeis cuja distância entre eles não seja maior que r unidades;

- σ_I e σ_X são hiperparâmetros definidos manualmente e controlam a importância que é dada à diferença de cores e à distância entre os píxeis, respetivamente. Estes hiperparâmetros foram definidos com os valores 10 e 4, respetivamente, não tendo sido alterados posteriormente. Estes valores foram os utilizados no artigo da W-Net e não sofreram alterações nas experiências efetuadas neste trabalho.

Ao treinar o U_{Enc} para minimizar a perda $J_{soft-Ncut}$ é possível, simultaneamente, minimizar a dissociação normalizada total entre os grupos e maximizar a associação normalizada total dentro dos grupos.

Resumidamente, a perda de corte normalizada suave é uma técnica que tem como objetivo criar uma representação no codificador composta por várias regiões. Essas regiões são formadas após segmentar a imagem original e correspondem aos diferentes objetos presentes na imagem ou zonas distintas da mesma.

2.1.5.3 Função de Perda de Reconstrução

De forma a garantir que as representações codificadas contenham o maior número de informações dos inputs originais, a rede W-Net é treinada para minimizar a perda de reconstrução. Ao minimizar esta perda, é possível fazer com que a previsão de segmentação se alinhe melhor com as imagens de input. A perda de reconstrução é dada pela Fórmula 2.12.

$$J_{reconstr} = \| X - U_{Dec}(U_{Enc}(X; W_{Enc}); W_{Dec}) \|_2^2 \tag{2.12}$$

Onde W_{Enc} representa os parâmetros do codificador, W_{Dec} representa os parâmetros do descodificador e X é a imagem de input. A rede W-Net é treinada de forma a minimizar $J_{reconstr}$ entre as imagens reconstruídas e os inputs originais.

Ao aplicar, iterativamente, $J_{reconstr}$ e $J_{soft-Ncut}$, a rede consegue equilibrar o trade-off entre a precisão de reconstrução e a consistência na camada de representação codificada.

2.1.5.4 Pós-Processamento

Após obter a segmentação inicial do codificador, são realizados dois passos de pós processamento de forma a obter a segmentação final.

Campos aleatórios condicionais totalmente conectados para recuperação de aresta precisa

As redes neurais convolucionais profundas com camadas de Max Pooling têm a capacidade de capturar as informações de features de alto nível do input. No entanto, o aumento da invariância e dos grandes campos recetivos podem causar uma redução da localização precisa. Por consequente, leva a uma falta de restrições de suavidade o que pode resultar num problema de delineamento deficiente do objeto, especialmente em tarefas de rotulação ao nível do píxel (Figura 2.14(b)).

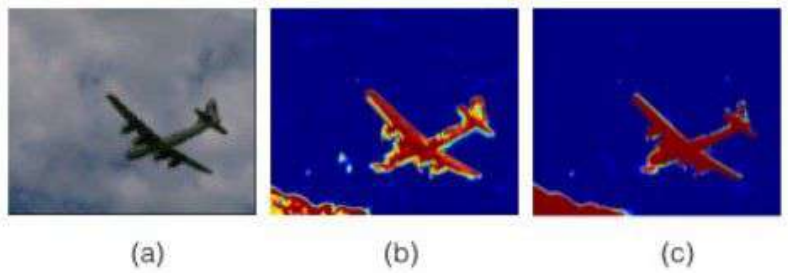


Figura 2.14: (a) Imagem Original, (b) Output da última camada do U_enc, (c) O output do CRF [50]

A função de perda do corte normalizado suave e as camadas de skip connection da rede W-Net podem ajudar a melhorar a localização desses limites, reduzindo o problema acima mencionados. Todavia, uma forma de melhorar o delineamento dos objetos consiste em combinar as respostas na camada U_{Enc} final com um modelo de campo aleatório condicional totalmente conectado (CFR) - Figura 2.14(c). Este modelo utiliza a Função de Energia 2.13.

$$E(X) = \sum_{u} \Phi(u) + \sum_{u,v} \Psi(u,v) \tag{2.13}$$

Onde:

- u e v são os pixéis da imagem de input X;
- **Potencial Unitário**: $\Phi(u) = -logp(u)$ onde p(u) é a probabilidade do píxel u pertencer à classe calculada pela camada de softmax do codificador;
- **Potencial Pairwise**: $\Psi(u,v)$ mede as penalidades ponderadas quando dois pixéis são atribuídos a diferentes rótulos. Para isso, utiliza dois kernels gaussian em diferentes espaços de features, podendo ser calculado da seguinte forma:

$$\Psi(u,v) = \mu(u,v)\left[w_1 exp\left(-\frac{\|P(u)-P(v)\|^2}{2\sigma_\alpha^2} - \frac{\|I(u)-I(v)\|^2}{2\sigma_\beta^2}\right) + w_2 exp\left(-\frac{\|P(u)-P(v)\|^2}{2\sigma_\gamma^2}\right)\right] \tag{2.14}$$

Onde $\mu(u,v) = 1$ se u $\neq$ v, 0 caso contrário. O que significa que apenas são penalizados os rótulos diferentes. O resto da expressão utiliza dois kernels gaussianos em diferentes espaços de características:

1. **Kernel Bilateral**: depende quer da posição dos píxeis (representado por P) quer da cor RGB (representado por I), forçando, assim, os pixéis com cores e posições semelhantes a terem rótulos semelhantes;
2. **Kernel**: depende apenas da posição dos píxeis, forçando, assim, apenas a suavidade.

Os hiperparâmetros σ_α, σ_β e σ_γ controlam as escalas dos Kernels Gaussianos.

Segmentação hierárquica

A segmentação hierárquica [2] [50] é construída com base em limites ponderados calculados a partir da importância de cada píxel nos limites da partição demasiado segmentada. É composta por duas fases:

- **Oriented Watershed Transform (OWT)**: construi uma região demasiado segmentada, criando um conjunto de regiões iniciais a partir do detetor de contorno que é definido pela Fórmula 2.15.

$$gPb(x,y,\theta) = \sum_s \sum_i \beta_{i,s} G_{i,\sigma(i,s)}(x,y,\theta) + \gamma \cdot sPb(x,y,\theta) \tag{2.15}$$

Onde $G_{i,\sigma(i,s)}(x,y,\theta)$ mede as diferenças no histograma no canal i entre duas metades do disco com raio $\sigma(i,s)$ centrado em (x,y) e dividido por um diâmetro com ângulo θ; $sPb(x;y;\theta)$ é o componente "espectral" do detetor dos limites 2.16.

$$sPb(x,y,\theta) = \sum_{k=1}^{n} \frac{1}{\sqrt{\lambda_k}} \cdot \nabla_{\theta V k}(x,y) \tag{2.16}$$

- **Ultrametric Contour Map (UCM)**: algoritmo de fusão de regiões, que, a partir dos limites das regiões obtidas na fase anterior, irá juntar as regiões semelhantes. Os contornos codificados na segmentação hierárquica resultante retêm pesos com valor real indicando a probabilidade de serem um limite verdadeiro.

Oriented Watershed Transform (OWT)

Utilizando o sinal de contorno, é construído um conjunto de regiões que determinam o nível mais alto de detalhe considerado. Este conjunto de regiões fornece uma imagem demasiado segmentada (Figura 2.15(b)). Este processo é feito através do cálculo da resposta máxima do detetor do contorno sobre as orientações 2.17

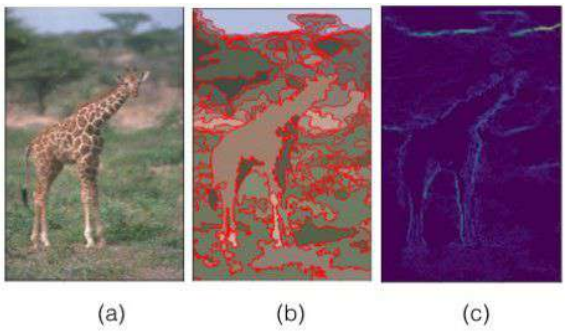


Figura 2.15: (a) Imagem Original, (b) Output (argmax) do CRF com limites a vermelho, (c) Mapas de limites ponderados correspondentes das linhas vermelhas em (b). [50]

$$E(x,y) = max_{\theta} E(x,y,\theta) \tag{2.17}$$

A simples ponderação de cada arco pelo valor médio de $E(x,y)$ para os píxeis no arco pode apresentar artefactos, provocado, principalmente, pela produção de uma resposta espacial estendida em torno dos limites fortes produzida pelo detetor de contorno. Uma forma de resolver esse problema é através da estimativa de uma orientação em cada píxel de um arco a partir da geometria local do próprio arco. Estas orientações são obtidas ao aproximar os arcos Watershed com segmentos de linha. Depois é subdividido, recursivamente, qualquer arco que não seja bem ajustado pelo segmento de linha conectando os pontos finais. Por último, é utilizado o output do detetor do contorno orientado $E(x,y,\theta)$ para atribuir cada píxel de arco (x,y) uma força de limite $E(x,y,o(x,y))$.

Ultrametric Contour Map (UCM)

A vantagem principal dos contornos é facilitar o processo de representação da incerteza da presença de um contorno subjacente verdadeiro. No entanto, não há uma forma imediata de representar a incerteza sobre uma dada segmentação. Uma possibilidade é a utilização do Ultrametric Contour Map (UCM), que define uma dualidade entre contornos ponderados fechados que não se intersetam e uma hierarquia de regiões. Os níveis superiores da hierarquia respeitam apenas contornos fortes, resultando numa segmentação mais reduzida. A hierárquica é construída por um algoritmo de fusão de regiões baseado num gráfico greedy.

O algoritmo classifica os links por similaridade e junta, iterativamente, as regiões mais semelhantes. Este processo produz uma árvore de regiões onde as folhas são os elementos iniciais das diferentes regiões, a raiz é a imagem completa e as regiões são ordenadas pela relação de inclusão.

Finalizando, esta arquitetura é bastante útil na segmentação de imagens de microscopia de bactérias porque não só fornece um mecanismo para segmentação de imagens de forma não supervisionada, mas também a aplicação de passos de pós-processamento permite definir, com bastante precisão, os limites dos objetos presentes na imagem, tornando os resultados deste método bastante bons quando comparados com outros métodos de segmentação supervisionada e não supervisionada. Desta forma, ao utilizar um método baseado na W-Net para a realização da segmentação das imagens de microscopia de bactérias, existe uma forte possibilidade em cumprir os dois principais objetivos deste trabalho: realização de segmentação não supervisionada e correção dos problemas que surgem quando temos grupos de bactérias juntas.

2.2 Funções de ativção

Como referido anteriormente, as funções de ativação têm o propósito de introduzir não linearidade no output da rede e algumas delas são apresentadas de seguida.

2.2.1 Função Linear

A função linear [1] é similar a uma linha direita, por exemplo: y = ax. Neste caso, não interessa o número de camadas utilizado, se todas forem lineares, o resultado irá ser linear, sendo utilizada apenas na camada de output.

2.2.2 Função Sigmoid

A função sigmoid [1] é utilizada na camada de output de uma classificação binária, onde o resultado é 0 ou 1, uma vez que o valor desta se encontra no intervalo [0, 1]. O resultado será igual a 1 se o valor da função for maior que 0.5 e 0 caso contrário.

2.2.3 Função Tangente Hiperbólica

Normalmente, a função tangente hiperbólica [1] tem melhores resultados que a função sigmoid e é utilizada nas camadas escondidas de uma rede neural pois os seus valores estão compreendidos entre -1 e 1 e, portanto, a média da camada escondida fica muito próxima de 0, ajudando a centralizar os dados, tornando assim, a aprendizagem para a próxima camada mais fácil.

2.2.4 ReLU

Normalmente, o ReLU é implementado nas camadas escondidas. É computacionalmente mais barata que a sigmoid ou a tangente pois envolve operações matemáticas mais simples. Para além [23] de ser computacionalmente mais eficiente, tem melhor performance e reduz o problema do Vanishing Gradient que ocorre quando as camadas mais baixas da

rede são treinadas demasiado devagar devido ao facto do gradiente diminuir exponencialmente ao longo das camadas.

2.2.5 Função Softmax

A função softmax [1] é do tipo da sigmoid, mas é utilizada no tratamento de problemas de classificação. Idealmente, é aplicada na camada de output para verificar se se está efetivamente a obter as probabilidades que definem a classe de cada input.

2.3 Funções de Perda

Nos últimos anos, foram propostos diversos algoritmos para segmentação de imagens que utilizavam diferentes funções de perda que são utilizadas durante o processo de aprendizagem.

2.3.1 Entropia Cruzada (Cross-Entropy)

A entropia cruzada [28] é derivada da divergência Kullback-Leibler (KL) que mede as diferenças em termos numéricos entre duas distribuições. Para tarefas comuns de aprendizagem automática, a distribuição dos dados é dada por um conjunto de treino. Assim, minimizar a divergência KL é equivalente a minimizar a entropia cruzada, que é definida por 2.18.

$$L_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} g_i^c \log s_i^c \tag{2.18}$$

Onde g_i^c é igual a 0 ou 1, indicando se o rótulo da classe c é a classificação correta para o píxel i, e s_i^c é a correspondente probabilidade prevista.

Uma extensão comum da Função apresentada em 2.18 é a entropia cruzada ponderada, que é dada por 2.19.

$$L_{WCE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} w_c g_i^c \log s_i^c \tag{2.19}$$

Onde w_c é o peso para cada classe que, normalmente, é inversamente proporcional às frequências das classes o que pode penalizar a maioria das classes.

Uma outra variação da entropia cruzada [6] é a entropia cruzada binária que pode ser descrita pela Fórmula apresentada em 2.20

$$L = -\frac{1}{f}\sum_{j=1}^{k} n_j \log m_j + (1-n_j)\log(1-m_j) \tag{2.20}$$

Onde f é o número de píxeis, k representa o número total de imagens utilizadas durante a fase de treino, n_j representa o valor original que é recebido do input e m_j o valor do output, ou seja o valor do descodificador.

Outras variantes desta função de perda são [28]: **perda topK**: tem como objetivo forçar a rede a focar-se nos exemplos mais difíceis durante o treino; **perda focal**: adapta a entropia cruzada padrão de forma a lidar com o desequilíbrio extremo das classes de primeiro plano que costumam ser sub-representadas, onde a perda atribuída a exemplos bem classificados é reduzida; **perda mapa de distância penalizando a entropia cruzada**: tem como objetivo orientar o foco da rede para regiões da fronteira que são difíceis de segmentar.

2.3.2 Dice Loss

A função de perda dice [28] tem a capacidade de otimizar diretamente o coeficiente dice que é a métrica mais utilizada para avaliação de segmentação. Ao contrário da entropia cruzada ponderada (weighted cross entropy), esta não requer uma nova ponderação da classe para tarefas de segmentação desequilibradas, que é dada pela Fórmula 2.21

$$L_{Dice} = \frac{2 * |X \cap Y|}{|X| + |Y|} \tag{2.21}$$

Onde $X \cap Y$ representa a interseção entre a imagem real X e a imagem segmentada Y; $|X|$ e $|Y|$ representa a cardinalidade de X e Y, respetivamente.

Apesar de ser bastante utilizado, o Dice Loss é uma função discreta, pelo que, com máscaras de segmentação, funciona quando temos a máscara de segmentação real e a criada pela nossa rede.

Existem outras funções de perda baseadas na função de perda dice: **perda de especificidade de sensibilidade** que aborda problemas desequilibrados, aumentando a especificidade; **perda IoU**, similar à perda dice, também é utilizada para otimizar, diretamente, a métrica de segmentação da categoria do objeto; **perda Tversky** que ao remodelar a perda dice e enfatizar os falsos negativos, alcança um melhor trade-off entre a precisão e recall; **perda dice generalizada** é uma extensão multi classe da perda dice onde o peso de cada classe é inversamente proporcional às frequências dos rótulos; **perda focal Tversky** aplica o conceito de perda focal a casos difíceis com baixa probabilidade; **perda de penalidade** penaliza falsos positivos e falsos negativos na função de perda dice generalizada.

2.3.3 Erro quadrático médio

O erro quadrático médio é uma das funções de perda mais utilizada para modelos de regressão e é a média dos quadrados das diferenças entre os valores originais e os valores previstos. Pode ser representada pela Fórmula 2.22

$$MSE = \frac{1}{N} \sum_{j=1}^{N} (predicted - input)^2 \tag{2.22}$$

2.4 Optimizadores

Todos os modelos de aprendizagem profunda e automática têm uma função de perda que avalia o desempenho do modelo num dado instante. O processo de aprendizagem caracteriza-se por utilizar essa função para treinar a rede de forma que esta tenha o melhor desempenho possível. Na prática, a função de perda é minimizada pois um erro menor significa um melhor desempenho do modelo. O processo de minimizar (ou maximizar) qualquer expressão matemática é denominada de otimização. Os optimizadores são algoritmos ou métodos utilizados para modificar os atributos da rede neural, com pesos e taxa de aprendizagem, de forma a diminuir as perdas, sendo utilizados para resolver problemas de otimização minimizando a função.

Existem diversos optimizadores como: Gradiente Descendente, Gradiente Descendente em Lote, SGD, SGD com momentum, RMSprop, Adagrad, Adam, entre outros.

2.4.1 Gradiente Descendente

O gradiente descente [40] é uma forma de minimizar uma função objetivo $J(\theta)$ parametrizada pelos parâmetros de um modelo $\theta \in R^d$ atualizando os parâmetros na direção oposta do gradiente da função objetiva $\nabla\theta J(\theta)$ com respeito aos parâmetros. O learning rate η determina o tamanho dos passos que são dados para atingir um mínimo local. Por outras palavras, a direção do declive da superfície criada pela função objetivo é seguida de cima para baixo até atingirmos o valor mínimo. De seguida, são apresentadas três variações do gradiente descendente.

2.4.1.1 Gradiente Descendente em Lote

Calcula [40] o gradiente da função de perda com respeito aos parâmetros θ de todo o conjunto de dados (Fórmula 2.23).

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.23}$$

Uma vez que é necessário calcular os gradientes para o conjunto de dados completo para realizar apenas uma atualização, o gradiente descendente em lote pode ser bastante lento e não se consegue utilizar com conjuntos de dados que não cabem na memoria.

2.4.1.2 Gradiente Descendente Estocástico

O Gradiente Descendente Estocástico (SGD) realiza uma atualização de parâmetro para cada exemplo de treino x^i e para cada rotulo y^i, sendo caracterizado pela Fórmula 2.24

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.24}$$

O Gradiente Descendente em Lote [40] realiza computações redundantes em conjuntos de dados grandes, uma vez que recalcula os gradientes para os exemplos parecidos

antes de cada atualização do parâmetro. O SGD, por sua vez, remove esta redundância ao realizar uma atualização de cada vez, o que, geralmente, o torna bastante mais rápido. Este optimizador realiza atualizações frequentes com uma elevada variação o que faz com que a função objetivo flutue bastante.

Ao contrário do Gradiente Descendente em Lote que converge para o mínimo local em que os parâmetros são colocados, a flutuação do SGD permite que este salte para mínimos locais novos e potencialmente melhores. Por outro lado, isso acaba por complicar a convergência para o mínimo local, já que o SGD continuará a ultrapassá-lo. No entanto, foi comprovado que quando se diminuí lentamente a taxa de aprendizagem, o SGD mostra o mesmo comportamento de convergência que o Gradiente Descendente em Lote.

2.4.2 SGD com Momentum

O Gradiente Descendente Estocástico tem problemas em áreas onde a superfície se curva muito mais abruptamente numa dimensão do que na outra, o que é comum em torno de locais ótimos. Nesses casos, este optimizador oscila ao longo de valores próximos desses mínimos locais enquanto faz um progresso hesitante ao longo do fundo em direção ao ótimo local.

O SGD com Momentum [36] é um método que ajuda a acelerar o SGD na direção relevante e amortece as oscilações. Para tal, adiciona uma fração γ do vetor de atualização do passo de tempo anterior ao vetor de atualização atual (Equações 2.25 e 2.26).

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \tag{2.25}$$

$$\theta = \theta - v_t \tag{2.26}$$

Analogamente, a utilização de momentum é como empurrar uma bola por uma colina abaixo. A bola acumula velocidade enquanto desce a colina, tornando-se cada vez mais rápida pelo caminho até atingir a sua velocidade terminal. Acontece o mesmo com a atualização do parâmetro: o termo momentum aumenta para as dimensões cujos gradientes apontam para a mesma direção e reduz as atualizações para as dimensões cujos gradientes mudam de direção. Como resultado, a convergência torna-se mais rápida e as oscilações são reduzidas.

2.4.3 Adagrad

Adagrad [9] é um algoritmo para otimizações baseadas em gradientes que adapta a taxa de aprendizagem aos parâmetros, realizando pequenas atualizações (i.e. taxas de aprendizagens baixas) para parâmetros associados com features que ocorrem com frequência e atualizações grandes (i.e. taxas de aprendizagens elevadas) para parâmetros associados a features pouco frequentes. Por esta razão é adequado para dados dispersos.

Um dos principais benefícios do Adagrad é que elimina a necessidade de ajustar manualmente a taxa de aprendizagem. A maioria das implementações utiliza o valor por defeito igual a 0.01 e é deixado com esse valor.

O principal problema do Adagrad é a acumulação de gradientes quadrados no denominador: como cada termo adicionado é positivo, a soma acumulada continua a crescer durante o treino. Isto faz com que a taxa de aprendizagem diminua, chegando a infinitesimalmente pequena, ponto em que o algoritmo se torna incapaz de aprender algo novo.

2.4.4 RMSprop

RMSprop foi proposto por Geoff Hinton em [44], tendo sido desenvolvido devido à necessidade de resolver o problema da grande diminuição da taxa de aprendizagem do Adagrad:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2 \tag{2.27}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t}} g_t \tag{2.28}$$

RMSprop divide a taxa de aprendizagem por uma média exponencialmente decrescente de gradientes quadrados. Hinton sugere que γ seja igual a 0.9 enquanto um bom valor para a taxa de aprendizagem seja 0.001.

2.4.5 Adam

Adaptative Moment Estimation (Adam) [21] é um modelo que calcula taxas de aprendizagem adaptativas para cada parâmetro. Para além de guardar uma média exponencialmente decrescente dos gradientes quadrados passados v_t, este optimizador também guarda uma média exponencialmente decrescente de gradientes passados m_t, semelhante ao SGD com momentum. Enquanto o SGD momentum pode ser visto como uma bola a descer uma colina, Adam pode ser visto como uma bola com atrito, que escolhe mínimos planos na superfície de erro. As médias decrescentes passadas (m_t) e os gradientes quadrados passados (v_t) podem ser obtidos ao utilizar as Fórmulas 2.29 e 2.30, respetivamente.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.29}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.30}$$

m_t e v_t são estimativas do primeiro momento (a média) e do segundo momento (a variância descentrada), respetivamente. Como m_t e v_t são inicializados como vetores de 0's, os autores de Adam observaram que eles tendem para zero, especialmente durante as etapas de iniciais e quando as taxas de decaimento são pequenas (i.e. β_1 e β_2 são próximas

de 1). Para neutralizar essas tendências, são calculadas as estimativas corrigidas pelo bias do primeiro e segundo momento:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{2.31}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{2.32}$$

As fórmulas 2.31 e 2.32 são utilizadas para atualizar os parâmetros, o que nos leva à seguinte regra de atualização do Adam (2.33).

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{(\hat{v}_t)} + \epsilon}\hat{m}_t \tag{2.33}$$

2.5 Bibliotecas Python

A linguagem de programação que irá ser utilizada na resolução do problema será o Python devido ao seu vasto conjunto de bibliotecas, principalmente as que se focam em aprendizagem automática e profunda. De seguida, são apresentadas algumas dessas bibliotecas.

2.5.1 Tensorflow

O Tensorflow [5] é uma biblioteca do Python para cálculos numéricas rápidas e pode ser utilizado na criação de modelos de aprendizagem profunda. Hoje em dia, existem outras bibliotecas que envolvem o Tensorflow também utilizadas na criação desses modelos, tornando o processo produzido no TensorFlow mais simples.

2.5.2 Keras

Keras [30] foi desenvolvida por François Chollet e é uma API de aprendizagem profunda de alto nível que permite a construção, treino, avaliação e execução de todos os tipos de redes neurais.

O keras começou a ser utilizado por outras bibliotecas, como é o caso do Tensorflow. Este tem uma versão com a sua própria implementação do keras (tf.keras) que, apesar de apenas suportar como backend o Tensorflow, tem a vantagem de oferecer algumas características extras como, por exemplo, suportar a API de dados do Tensorflow o que facilita o carregamento e realiza o pré-processamento dos dados de forma eficiente. Por esta razão, esta versão do keras foi utilizada neste trabalho.

O tf.keras possui diversos componentes utilizados na construção de modelos de aprendizagem profunda como os autoencoders, dos quais se destacam-se os seguintes:

- **Otimizadores**: são passados como parâmetros quando se faz compile do modelo. Os optimizadores mais frequentes são: **adam**, caracterizado por ser um método de

descida do gradiente estocástico, baseando-se na estimativa adaptativa de momentus de primeira e segunda ordem; e **SGD** é um optimizador de descida do gradiente com momentum.

- **Funções de Perda**: como referido anteriormente, existem diversas funções de perda que podem ser utilizadas para treinar o autoencoder. Da mesma forma que os optimizadores, estas também são passadas como parâmetro na chamada compile. O tf.keras contêm várias funções, onde se destacam as seguintes: cross-entropy, erro quadrático médio, poisson.

- **Funções de Ativação**: o tf.keras fornece diversas funções de ativação incluindo as mais utilizadas: linear, relu, sigmoid e softmax.

2.5.3 Numpy

A biblioteca NumPy [46] compreende não só a matriz NumPy mas também um conjunto de funções matemáticas que a acompanham. Um array NumPy é uma coleção multidimensional e uniforme de elementos, ou seja, todos os elementos ocupam o mesmo número de bytes na memória. Esta biblioteca é bastante utilizada porque apesar das listas em Python poderem ser manipuladas como arrays, o processamento destas é bastante lento. O objetivo do NumPy é fornecer um objeto do tipo array bastante mais rápido que as tradicionais listas em Python.

2.5.4 Matplotlib

Matplotlib [17] é um pacote de gráficos 2D utilizado no Python para o desenvolvimento de aplicações, scripts interativos e geração de imagens com qualidade de publicação em interfaces de utilizadores e sistemas operativos. Para além disso, oferece uma forma de criar gráficos de alta qualidade para os principais tipos de gráficos 2D, como gráficos de dispersão ou gráficos de barras.

2.5.5 Scikit-Learn

Scikit-learn [32] é uma das bibliotecas mais populares de aprendizagem automática, sendo construída utilizando as bibliotecas numpy, scipy e matplotlib, focando-se na modelagem de dados ao invés de carregar, manipular, visualizar e resumir dados.

Scikit-learn [34] é um módulo do Python que integra uma vasta gama de algoritmos de aprendizagem automática para problemas supervisionados e não supervisionados de média escala. Assim, esta biblioteca fornece implementações de muitos algoritmos de aprendizagem automática da literatura, mantendo uma interface fácil de utilização totalmente integrada com o Python.

2.6 Considerações Finais

Recentemente [26], métodos baseados em aprendizagem profunda ganharam uma popularidade crescente devido à sua precisão elevada e à melhor generalização. É o caso da arquitetura U-Net que foi proposta em 2015 e realiza segmentação supervisionada de imagens. No entanto, segmentação supervisionada de imagens de microscopia de bactérias, como referido anteriormente, requer uma grande quantidade de imagens manualmente segmentadas, o que requer bastante tempo. Por esta razão, a rede U-Net não será utilizada neste trabalho. Apesar disso, esta arquitetura esteve na base de muitos algoritmos de segmentação quer fosse supervisionada ou não supervisionada. É assim que surge a arquitetura W-Net que permite a segmentação não supervisionada de imagens que, com dois passos de pós-processamento, obtém uma imagem segmentada com um erro bastante baixo. Esta rede caracteriza-se por duas redes semelhantes à U-Net que, juntas, realizam a segmentação não supervisionada de imagens. Assim sendo, a primeira rede totalmente convolucional funciona como codificador codificando a imagem de input numa segmentação k-way utilizando, para isso, camadas totalmente convolucionais. A segunda rede totalmente convolucional reverte o processo feito pela primeira rede, reconstruindo a imagem de input original a partir da camada de segmentação. Durante esse processo, de forma conjunta, é realizado a minimização quer da função de perda de reconstrução do autoencoder quer da função de perda do corte normalizado suave da camada de codificação. Após este processo, a imagem segmentada contém alguns problemas como a presença de pequenas manchas ou a segmentação em demasia. Para correção desses problemas, são aplicados dois passos de pós-processamento: o primeiro é aplicação de um campo aleatório condicional totalmente conectado que torna a imagem de segmentação mais suave e o segundo passo é a aplicação do método da segmentação hierárquica para obtenção da segmentação final.

Para realizar segmentação não supervisionada de imagens de microscopia de bactérias foi implementada a arquitetura W-Net descrita no parágrafo anterior e proposta por Xide Xia e Brian Kulis em [50] mas, com algumas modificações. Primeiro, foi implementada uma forma de alterar os filtros das camadas convolucionais de forma a ser possível modificar facilmente a profundidade de ambas as redes que compõem a W-Net. Esta medida permitiu uma melhor adaptação da rede aos conjuntos de dados utilizados.

Segundo, os passos de pós-processamento utilizados em [50] não foram utilizados. Em vez disso, o primeiro passo passou por combinar as classes do output da rede codificadora para encontrar as melhores máscaras de segmentação. De seguida, a essas máscaras de segmentação foi aplicado o algoritmo de segmentação Watershed para obtenção da máscara de segmentação final. Todas estas alterações têm o objetivo de otimizar a rede W-Net e testar se é possível utilizá-la para segmentação de imagens de microscopia de bactérias. Adicionalmente, com a implementação da rede W-Net, espera-se conseguir resolver ambos os problemas mencionados anteriormente: os erros ocorridos durante o processo de segmentação de grupos de bactérias juntas e a necessidade de grandes quantidades de

dados manualmente rotulados.

Os autoencoders, como a W-Net, possuem as seguintes vantagens: 1) A profundidade do autoencoder pode reduzir exponencialmente não só o custo computacional de representar algumas funções, mas também a quantidade de dados de treino para aprender algumas funções; 2) Um autoencoder permite a aprendizagem não supervisionada o que é bastante útil na segmentação de imagens de microscopia onde a falta de dados manualmente rotulados é um problema na segmentação supervisionada; e 3) As redes neurais convolucionais têm uma grande capacidade em aprender a partir de padrões, características ou texturas em vez de se basear nas variações de cor.

3

Trabalho Experimental

Como em qualquer modelo de aprendizagem, é necessário um conjunto de dados para treiná-lo. Todos os conjuntos de dados utilizados para treinar a rede desenvolvida não continham qualquer tipo de rótulo, o que permitiu treinar a W-Net sem a necessidade de se identificar manualmente cada elemento da imagem ou de aplicar um passo de pré-processamento para realizar essa identificação.

Numa primeira fase, serão apresentados os conjuntos de dados utilizados no desenvolvimento desta dissertação. Adicionalmente, será explicado o tratamento realizado aos diferentes conjuntos de dados antes de serem utilizados no processo de treino.

De seguida, será apresentada uma breve descrição da implementação da rede bem como o processo completo de obtenção da máscara de segmentação. Por último, serão apresentadas as experiências efetuadas, os resultados obtidos e uma breve discussão para a obtenção dos melhores resultados para os diferentes parâmetros da rede.

3.1 Conjunto de Dados

Foram utilizados dois grupos de imagens distintas durante o processo de treino: imagens de pokémons e de bactérias. O primeiro é um conjunto de imagens artificiais que contém não só as imagens de treino, mas também as máscaras de segmentação. A utilização deste conjunto teve como principal objetivo facilitar a explicação do processo efetuado para obter as máscaras de segmentação das bactérias. Adicionalmente, teve como objetivo a validação do processo de criação das máscaras de segmentação. Já o segundo conjunto de dados é composto por imagens de microscopia de bactérias, sendo o conjunto de dados principal desta dissertação.

3.1.1 Conjunto de Dados: Imagens Pokémons

Este conjunto de imagens é composto por imagens de pokémons encontrados perto do campus da FCT-UNL (Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa). Estas imagens foram obtidas ao sobrepor, de forma aleatória, as imagens dos pokémons em fotografias tiradas do Google Street View. Cada imagem é definida por

um array de 64x64x3 normalizado para valores compreendidos no intervalo [0,1], com 64x64 píxeis e três canais de cores (RGB). Um exemplo dessas imagens está presente na Figura 3.1.

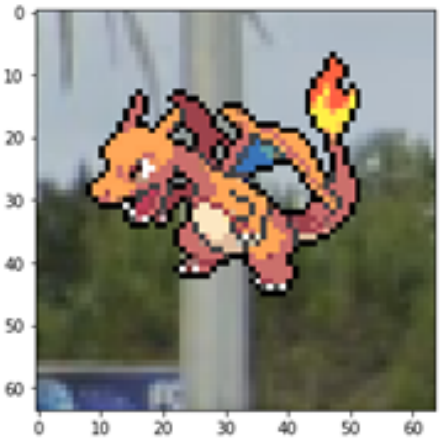


Figura 3.1: Exemplo Imagem de Treino Pokémon

Este conjunto de dados foi dividido nos seguintes subconjuntos: 3500 imagens para treino, 500 imagens para validação e 500 imagens para testar os resultados obtidos.

Como referido anteriormente, este conjunto de dados é artificial sendo utilizado para fins demonstrativos do que se pretende realizar com as imagens das bactérias, tornando-se mais fácil por termos acesso às máscaras de segmentação correspondentes. Adicionalmente, foi utilizado para validação do processo de criação das máscaras de segmentação. Por último, foi ainda utilizado no processo de escolha dos valores dos diferentes parâmetros em casos em que a rede não convergia com as opções selecionadas.

3.1.2 Conjunto de Dados: Imagens Bactérias

As imagens de bactérias utilizadas para treinar a rede desenvolvida foram cedidas pelo Bacterial Cell Biology Lab do ITQB (https://www.itqb.unl.pt/research/biology/bacterial-cell-biology). Estas foram criadas a partir dos TIFFs (Tagged Image File Format – formato de arquivo para imagens digitais) originais e, à semelhança das imagens dos pokémons, a escala destas imagens foi normalizada para estar compreendida no intervalo [0,1].

Estas imagens encontram-se divididas em três grupos distintos: **Phase**, **Vermelho Texas** e **DAPI** (4',6-diamidino-2-phenylindole). O conjunto **Phase** consiste num conjunto de fotografias com contraste de fase que marca a silhueta das células. Este tipo de contraste é uma técnica utilizada para obter o contraste numa amostra translucida sem manchar a mesma. A Figura 3.2 mostra um exemplo deste tipo de imagens.

Vermelho Texas consiste num corante fluorescente vermelho que é utilizado na coloração de células. Neste caso, este corante está ligado a proteínas da membrana, dando mais brilho às membranas das células. A Figura 3.3 é um exemplo onde se aplicou este corante.

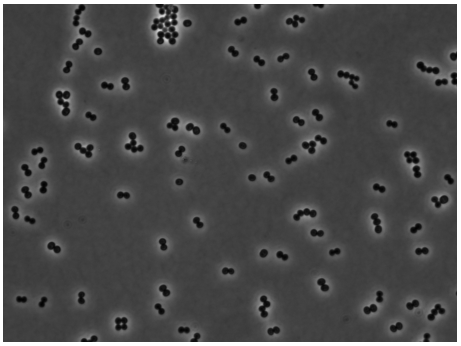


Figura 3.2: Exemplo Imagem Phase

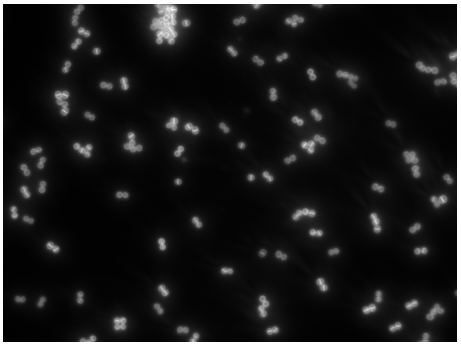


Figura 3.3: Exemplo Imagem Vermelho Texas

Por último, o **DAPI** é um marcador fluorescente com a capacidade de se ligar ao ADN. Ao conectar-se às regiões adenina-timina do ADN consegue emitir fluorescência, o que permite que parte do cromossoma das bactérias fique visível. Um exemplo da aplicação deste marcador encontra-se apresentado na Figura 3.4.

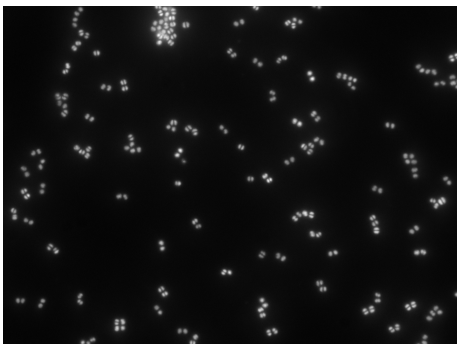


Figura 3.4: Exemplo Imagem DAPI

Tal como uma imagem colorida tem três canais distintos que correspondem às cores primárias vermelho, verde e azul, os três conjuntos de dados das bactérias acima mencionados podem ser vistos como três canais distintos que juntos formam um conjunto final para treinar a rede. Isto é possível pois cada imagem de um conjunto tem uma correspondente nos restantes conjuntos. Para juntar os três conjuntos aplicou-se um passo de pré-processamento às imagens dos conjuntos Phase, Vermelho Texas e DAPI. Este passo consistiu no empilhamento das imagens correspondentes de cada um dos conjuntos pela seguinte ordem Phase, Vermelho Texas e DAPI, resultado um único conjunto de dados.

Para isso, utilizou-se a função do Numpy: dstack que se caracteriza por empilhar matrizes ao longo do terceiro eixo. O conjunto de dados resultante será denominado por all_channels. A representação das imagens pertencentes a este conjunto de dados está a ser feito através de uma imagem RGB, onde o canal Red corresponde ao canal Phase, o Green corresponde ao canal Vermelho Texas e o último canal corresponde ao DAPI. Apesar dos canais Phase, Vermelho Texas e Dapi não serem canais de cores, optou-se por esta forma para simplificar o processo de visualização do conjunto criado.

Por último, ainda no pré-processamento, todas as imagens do conjunto de dados all_channels bem como as pertencentes aos três canais foram recortadas devido a limitação de hardware. O tamanho inicial de cada imagem correspondia a 1024x1376 e, para cada uma, procedeu-se ao seu recorte de forma a possuírem a seguinte dimensão: 256x256. Para isso, teve de se ter em atenção que se estava a recortar uma imagem não quadrada em várias mais pequenas quadradas. Para garantir que as imagens recortadas resultantes continham todos os píxeis da imagem inicial, verificou-se se seria possível dividi-la em n imagens de 256x256. Para isso, procedeu-se à divisão da altura e largura por 256 e concluiu-se que se poderia dividir a altura em 4 pedaços de 256 cada. Mas, o mesmo não aconteceu para a largura visto que o resultado da divisão não ser um número inteiro (5,375). Se o resultado fosse arredondado para 5, perder-se-ia 96 (1376-5*256) colunas de píxeis. Por outro lado, se o valor fosse arredondado para 6, era necessário acrescentar 256*6-1376 = 160 colunas de píxeis dummies para se obter uma imagem de 256x256. Assim, de forma a não ter de se criar píxeis dummies e para considerar todos os píxeis da imagem inicial, começou-se por realizar a divisão da parte da imagem que não continha qualquer problema: ou seja desde o ponto (0,0) da imagem (canto superior esquerdo da imagem) até ao ponto (1024, 1280) da imagem (canto inferior direito da imagem menos os 96 píxeis "a mais"). De seguida, para também se ter acesso ao conjunto de píxeis das últimas colunas que não foram considerados até ao momento, procedeu-se à criação de subimagens da região da imagem inicial que se encontrava dentro dos seguintes limites: (0, 1376-256) a (1024, 1376). Na Figura 3.5 encontra-se esquematizado o processo de recorte da imagem inicial implementado. Procedeu-se ao recorte das imagens e não ao seu redimensionamento porque a rede é totalmente convolucional e o tamanho selecionado (256x256) ainda preserva todos os detalhes que a rede precisa de aprender.

Apesar do conjunto utilizado nas experiências ser o all_channels, optou-se por guardar, num mesmo ficheiro, as imagens dos restantes canais porque poderiam ser utilizados em futuras experiências. Assim, os diferentes conjuntos já estariam preparados para treinar o modelo, sendo possível utilizar os seguintes conjuntos de treino durante o processo de treino: Phase, Vermelho Texas, DAPI e All_Channels (que corresponde aos três canais empilhados).

Para cada um dos conjuntos de dados das bactérias, é feito uma partição do mesmo de forma a ter acesso aos seguintes três grupos de imagens com imagens distintas: **conjunto de treino** (60% do conjunto inicial), **conjunto de validação** (20% do conjunto de dados) e **conjunto de teste** (20% do conjunto inicial).

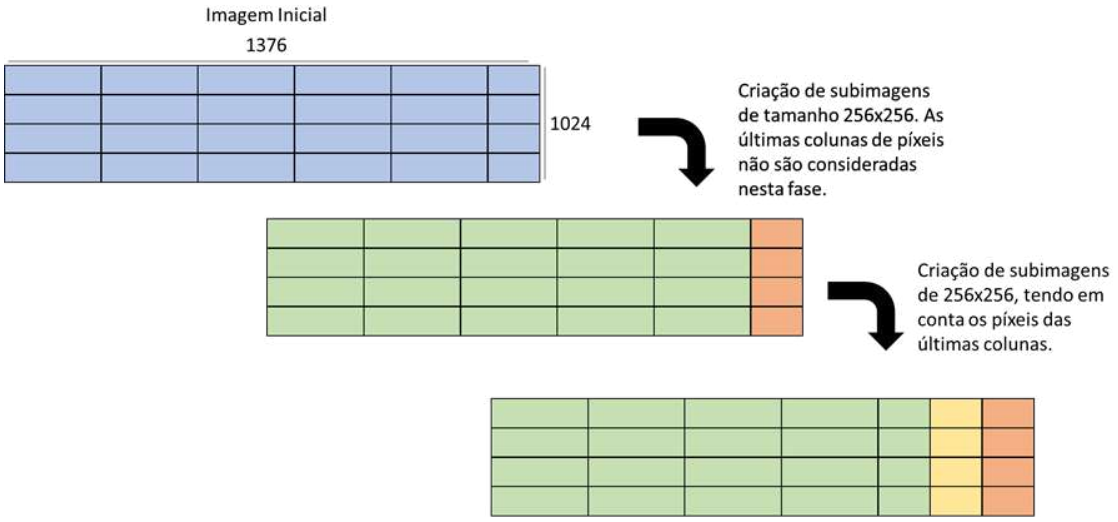


Figura 3.5: Esquema Recorte Imagem

3.2 Descrição da Rede Implementada

Para obter as máscaras de segmentação foi implementada uma versão da W-Net descrita em [50]. Na Figura 3.6 é apresentada um esquema desta mesma rede.

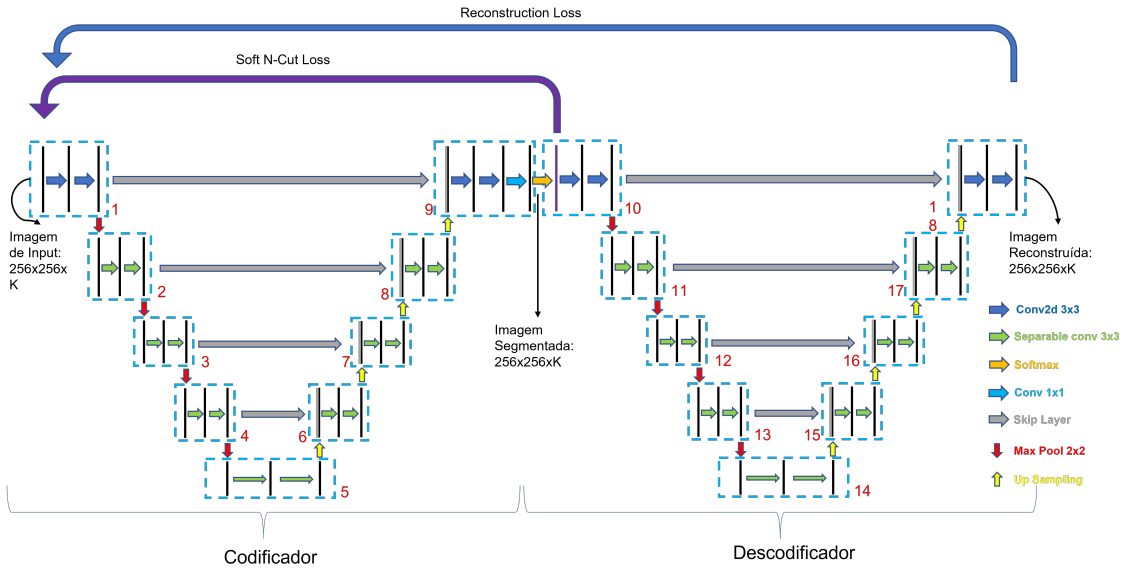


Figura 3.6: Exemplo Arquitetura W-Net (Baseada na Arquitetura apresentada em [50])

A arquitetura da rede implementada é composta pela rede codificadora e pela rede descodificadora. A primeira recebe uma imagem de tamanho 256x256x3 e devolve uma imagem segmentada de tamanho igual a 256x256xK, onde o K representa o número de classes pré-estabelecido. Para isso, a rede codificadora é composta por um caminho contrativo que irá capturar o contexto dos dados e por um caminho expansivo que irá ativar a localização precisa. Ambos os caminhos são compostos por módulos que são constituídos por: duas convoluções, cada uma seguida por ReLU e batch normalization. A única diferença entre ambos os caminhos é que, após o batch normalization, no contrativo

é aplicado Max Pooling e no expansivo é aplicado Up Sampling. A camada convolucional final da rede codificadora é uma convolução 1x1 seguida pela ativação softmax. Esta convolução irá mapear cada vetor de features para o número desejado de classes K e, de seguida, a camada softmax redimensiona-os para que os elementos do output K fiquem compreendidos entre 0 e 1 e cuja soma seja igual a 1.

Por outro lado, a rede descodificadora irá receber a imagem segmentada criada pela rede codificadora de tamanho igual a 256x256xK e vai tentar recriar a imagem inicial que foi introduzida na rede codificadora. À semelhança da primeira rede, esta também é composta por dois caminhos, o contrativo e o expansivo, e cada um deles é composto pelos mesmos módulos que a rede codificadora. A última camada da rede descodificadora é uma convolução 1x1 seguida pela ativação sigmoid pois este é um output de uma classificação.

A rede codificadora e a descodificadora que compõem a W-Net podem ser vistas como sendo compostas por um conjunto de módulos que são formados por duas convoluções, seguidas por batch normalization e Max Pooling (ou Up Sampling, dependendo do caminho em questão). Na Figura 3.6, encontra-se uma representação geral da rede, onde cada um dos retângulos azuis corresponde a um destes módulos. Aproveitando esta ideia de módulos, foi pensada na seguinte implementação da rede baseada no número de filtros que são utilizados nas camadas convolucionais de cada módulo. Esta implementação é caracterizada por receber um conjunto de valores que irão corresponder ao valor dos filtros utilizados em cada um dos módulos. Por exemplo, se for definido que a rede recebe os seguintes valores – 16, 32, 64, 128 e 512 – para os filtros, significa que as convoluções do primeiro módulo terão filtros de valor igual a 16, as convoluções do segundo módulo terão filtros com valor igual a 32 e por aí em diante. Mas, como é possível de se observar, o número total de valores fornecidos para os filtros é inferior ao número total de módulos presentes na Figura 3.6. Isto acontece porque os valores se repetem. Ou seja, após o módulo 5, que ficará com o valor 512, começa-se a utilizar os filtros de forma decrescente, sendo que as convoluções do módulo 6 ficam com filtros iguais a 128 e por aí em diante, funcionando como espelho. Na Figura 3.7, é apresentado um esquema da divisão completa destes valores pelos diferentes módulos, para o exemplo dado. É de notar que a criação dos módulos é dinâmica, dependendo do número de filtros que se definem à priori. Se se forem definidos apenas 3 filtros, a rede terá apenas uma profundidade igual a 3, sendo o terceiro módulo equivalente ao quinto módulo da Figura 3.6. Assim, é possível não só a alteração rápida dos valores dos filtros, mas, acima de tudo, é possível alterar a profundidade da rede com bastante facilidade.

As funções de perda dos modelos de aprendizagem profunda têm como objetivo avaliar as previsões que a rede efetua. No caso da W-Net implementada é minimizado a função de perda de reconstrução da rede completa bem como a função de perda do corte normalizado suave da camada codificadora.

Como explicado com mais detalhe na secção 2.1.5.3, a função de perda de reconstrução tem como objetivo garantir que as representações codificadas contenham o maior número de informações dos inputs originais. Esta função de perda pode ser calculada utilizando

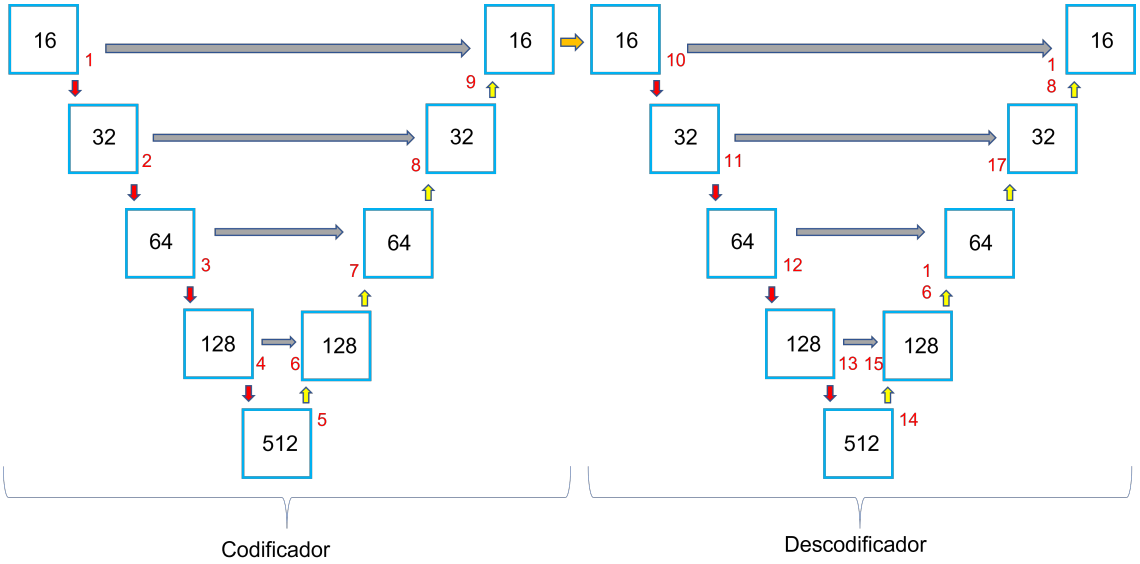


Figura 3.7: Aplicação dos filtros definidos

a Fórmula 3.1.

$$J_{reconstr} = \|X - U_{Dec}(U_{Enc}(X; W_{Enc}); W_{Dec})\|_2^2 \tag{3.1}$$

Onde X é a imagem inicial e W_{Enc} e W_{Dec} são os parâmetros da rede codificadora e descodificadora, respetivamente. Esta fórmula foi adaptada e utilizada nesta dissertação como se encontra na Fórmula 3.2.

$$J_{reconstr} = \|X - Y\|_2^2 \tag{3.2}$$

Onde o X representa a imagem inicial e o Y representa a imagem reconstruída pela rede descodificadora. Ou seja, a função de perda utilizada para medir a diferença entre o resultado da W-Net e a imagem inicial irá aplicar o quadrado da norma euclidiana à diferença entre a imagem inicial e a imagem recriada. A norma euclidiana é calculada utilizando a Expressão 3.3.

$$\|x\|_2 = \sqrt{x_1^2 + \ldots + x_n^2} \tag{3.3}$$

A função de perda do corte normalizado suave tem como objetivo criar uma segmentação no codificador composta por várias regiões. Essas regiões são formadas após segmentar a imagem original e correspondem aos diferentes objetos presentes na imagem ou zonas distintas da mesma. Este erro foi calculado utilizando a Fórmula 2.10.

Para ser possível treinar a rede codificadora para minimizar a função de perda do corte normalizado suave e, de seguida, treinar a rede completa para minimizar a função de perda de reconstrução, o processo de treino incluiu dois passos distintos: um primeiro onde se treina a rede codificadora minimizando a função de perda do corte normalizado suave e um segundo passo onde se treina a rede completa para minimizar a função de

perda de reconstrução. Para tal, utilizou-se um processo de treino semelhante ao da Rede Adversária Geradora (ou Generative Adversarial Network – GAN), onde o discriminador e o gerador são treinados de forma alternada e separada. No caso da rede W-Net implementada, o processo de treino é caracterizado por, em cada época, primeiro treina-se o modelo codificador de forma a minimizar a função de perda do corte normalizado suave e, de seguida, é treinada a rede W-Net como um todo (ou seja, quer o codificador quer o descodificador), minimizando a função de perda de reconstrução. Com exceção do grupo de experiências para obtenção do optimizador e taxa de aprendizagem, foi utilizado o mecanismo de early stopping no processo de treino. Este mecanismo caracteriza-se por o processo de treino continuar enquanto a validação da perda de reconstrução desce. Se esta validação não diminuir ao fim de 3 épocas seguidas o algoritmo termina. Caso esta situação não ocorra, ao fim de 200 épocas o processo de treino também é dado como terminado. O Algoritmo 1 apresenta, em pseudocódigo, o processo de treino aplicado ao modelo criado.

Algorithm 1: Processo de Treino W-Net

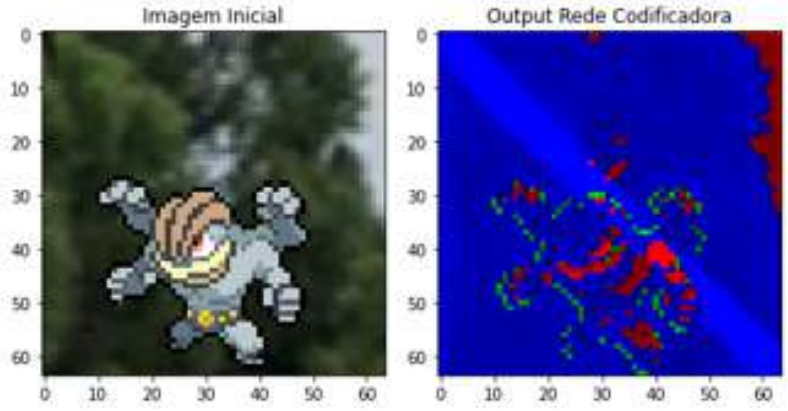
```
Definir o número de exemplos por cada época
Definir o half_batch
while ValidacaoErrodeReconstrucaoDesce do
  for j ∈ BatchPerEpoch do
    Obter subconjuntos de imagens para X, y
    Treinar o codificador com X e y
    Treinar a rede toda com X e y
  end for
end while
```

Onde:

- **BatchPerEpoch**: número de exemplos utilizados em cada iteração do modelo;
- **Half_batch**: número de elementos de X e y, que vai corresponder a metade do valor de batch;
- **X**: dados de input;
- **y**: dados alvo.

Resumidamente, para treinar a rede foi definido o número de exemplos a utilizar em cada época o que equivale à divisão entre o total de elementos do conjunto de dados e o tamanho do batch. Também foi definido o número de elementos do conjunto de dados a ser utilizado no conjunto de treino da época corrente que corresponde a metade do tamanho do batch. De seguida, foi criado um ciclo while que irá terminar quando se atingir uma das duas condições seguintes: o valor da validação da função de perda de reconstrução não descer durante três épocas consecutivas ou são atingidas as 200 épocas. Dentro do ciclo while, é definido um ciclo for que, para cada época, irá iterar sobre o

número de exemplos definido. No processo de treino de um modelo, é obrigatório definir dois conjuntos distintos: X e y. O primeiro refere-se ao conjunto que contém os dados para treinar a rede e o segundo contém os dados alvos para serem utilizados durante o treino. Todavia, em algoritmos não supervisionados apenas são utilizados dados sem rótulo e é obrigatório definir um conjunto y. Isto motiva a que ambos os conjuntos, X e y, sejam iguais em cada iteração do modelo em algoritmos não supervisionados. Assim, dentro do for começou-se por obter um subconjunto do conjunto inicial de dados (para o X e y) que será utilizado para treinar os modelos na iteração corrente. Para isso, recorreu-se à função do numpy random.randint para gerar, aleatoriamente, um conjunto de valores inteiros de tamanho igual a metade do tamanho do batch definido. Esses valores serão utilizados como índices para obter um subconjunto do conjunto completo de imagens que corresponderá ao conjunto X e y. De seguida, utilizando a função do Keras train_on_batch o codificador e o modelo completo são treinados com os dados em X e y. Depois de treinar a rede, é possível obter uma imagem segmentada (Figura 3.8(b)) a partir da rede codificadora, ao fornecer uma imagem inicial (Figura 3.8(a)).



(a) Imagem Inicial (b) 1: Vermelho, 2: Verde, 3: Azul

Figura 3.8: Exemplo Output Camada Codificadora

O número de classes (K) representa o número total de classes pelas quais se vão distribuir os píxeis da imagem e pode ser utilizado para encontrar as melhores máscaras de segmentação. Para isso, começa-se por calcular todas as combinações possíveis para o dado número de classes. Por exemplo, se a rede foi treinada utilizando três classes, obtêm-se as seguintes combinações: 1, 2, 3, 1+2, 1+3, 2+3 e 1+2+3. Cada uma destas combinações indicam as classes do output do codificador que serão utilizados para gerar a máscara de segmentação: no caso de se considerar a primeira combinação, estar-se-ia a utilizar apenas os valores correspondentes à primeira classe para obter a máscara de segmentação. Por outro lado, se considerarmos a combinação 1+2, a máscara de segmentação corresponderia à imagem resultante de somar os valores dos píxeis das classes 1 e 2. Como se está a somar as classes do resultado da aplicação do softmax, a soma de todas as probabilidades ou ativações de todas as classes, para cada píxel, encontra-se compreendida entre 0 e 1.

Assim, para cada combinação de classes possível, caso a soma das ativações dessas classes seja inferior a 0.5, a máscara de segmentação é 0 nesse ponto. Caso seja maior ou igual a 0.5, a máscara é igual a 1 nesse ponto. A Figura 3.9 é um exemplo ilustrativo utilizando as imagens dos pokémons daquilo que se pretende implementar com o conjunto de dados das bactérias.

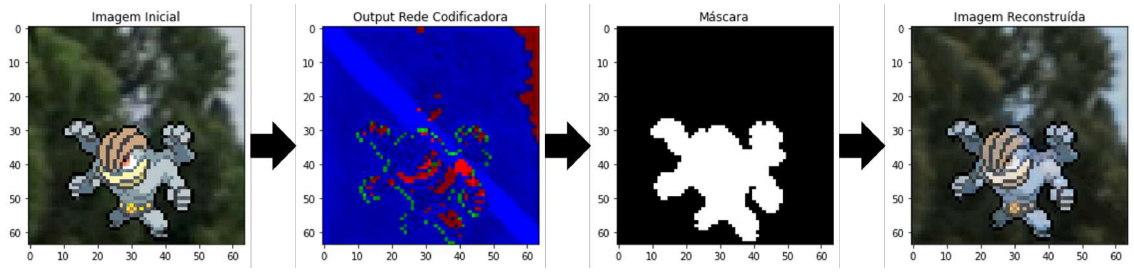


Figura 3.9: Exemplo Ilustrativo Processo Completo de Obtenção da Máscara de Segmentação

Resumidamente, utilizando a Figura 3.9 como exemplo, para criar a máscara de segmentação final é essencial realizar os seguintes passos: 1) Utilizar uma imagem inicial, após treinar a rede, para obter o output da rede codificadora que corresponde ao resultado da ativação softmax (imagem resultante da rede codificadora); 2) Calcular o número de combinações possíveis com o número de classes com que se treinou a rede; 3) Para cada combinação de classes obtida em 2, realizar a soma das classes respetivas, utilizando o resultado obtido em 1; 4) Para obter a máscara de segmentação final, para cada ponto do resultado da soma feita em 3 verificar se é maior ou igual 0.5. Se for, esse ponto passa a 1, caso contrário passa a 0.

Um exemplo do resultado obtido quando se utilizou o conjunto de dados dos pokémons para treinar a rede encontra-se apresentado na Figura 3.10. Aqui treinou-se a rede com um número de classes igual a 3. De seguida, obtiveram-se as todas as combinações de classes possíveis e as melhores máscaras de segmentação obtidas encontram-se na Figura 3.10.

As diferenças notórias entre as máscaras obtidas e a máscara de teste (Figura 3.9 - máscara) devem-se, principalmente, à dificuldade de segmentação deste tipo de imagens. As imagens de input, para além dos pokémons diferentes com formas e tamanhos diferentes, também têm um fundo detalhado e diferente entre as imagens. Isto faz com que o processo de treino se torne bastante mais difícil do que quando comparado com os conjuntos das bactérias onde o fundo é homogéneo e igual entre as diferentes imagens e as bactérias têm todas a mesma forma. Assim, pode-se concluir que o procedimento proposto funciona e que poderá ser utilizado na segmentação das bactérias.

3.3 Optimizações da Rede W-Net

Umas das otimizações implementadas foi a utilização de convoluções separáveis em profundidade, como proposto no artigo da W-Net ([50]). Este tipo de convoluções, como

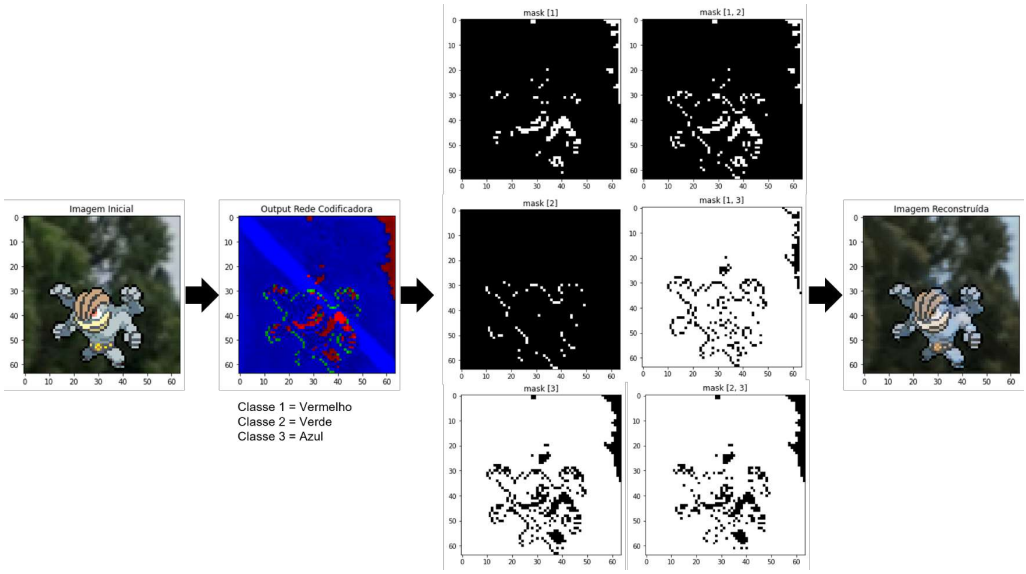


Figura 3.10: Número de Classes = 3 Pokémons

explicado na Secção 2.1, realizam uma convolução espacial em profundidade seguida por uma convolução pontual. A primeira convolução aplica um filtro que afeta apenas um canal de cada vez da imagem, aplicado a todos os canais da imagem. A segunda convolução tem como objetivo aumentar o número de canais da imagem resultante da primeira convolução. Com a utilização deste tipo de convolução nos módulos intermédios (utilizando a Figura 3.6 como exemplo, nos módulos 2 a 8 e 11 a 17), a rede é capaz de processar mais informação num menor período de tempo, pois o número de multiplicações que tem de efetuar é consideravelmente inferior ao número de multiplicações realizadas nas convoluções normais.

Por exemplo, se o tamanho inicial da imagem for 12x12x3 e se forem aplicados 6 kernels de tamanho igual a 5x5x3 com stride de 1 e sem padding, os filtros irão mover-se 8x8 vezes (12-5+1). Assim, numa convolução normal, irão existir 6x5x5x3x8x8 = 28 800 multiplicações. Já no caso das convoluções separáveis em profundidade, a parte da convolução em profundidade contém 3 filtros de tamanho 5x5x1 que se movem 8x8 vezes o que dá 3x5x5x8x8 = 4 800 multiplicações. A parte convolução pontual contém 6 filtros de 1x1x3 que se movem 8x8 vezes, o que equivale a 6x1x1x3x8x8 = 1 152 multiplicações. Ao efetuar a soma de ambas as multiplicações, é obtido um total de 5 952 multiplicações, valor bastante menor quando comparado com as 28 800 multiplicações da convolução normal. O principal fator para esta diferença é que na convolução normal, a imagem é transformada 6 vezes, sendo que cada transformação utiliza 5x5x3x8x8=4800 multiplicações. Por outro lado, na convolução separável em profundidade, a imagem é apenas transformada uma única vez – na convolução espacial em profundidade. Depois, essa imagem é apenas alongada para os 6 canais. Adicionalmente, este tipo de convoluções ajudam a examinar as corelações espaciais e as correlações entre canais de forma independente.

Como explicado na secção 3.2, a W-Net foi implementada de forma que a rede possa ser representada por módulos, como apresentado pelos retângulos azuis na Figura 3.6.

Cada módulo é composto por duas convoluções, cada uma seguida por ReLU e por batch normalization e Max Pooling ou Up Sampling. A implementação da arquitetura W-Net baseada em módulos e no número de filtros permitiu criar um modelo em que se altera a profundidade da rede com relativa facilidade. Isto ocorre porque a profundidade encontra-se associada ao número de módulos, o que por sua vez, está associado ao número de filtros, valores estes que são alterados facilmente. À alteração da profundidade da rede está associada uma variação do número de Down Sampling e Up Sampling aplicados à imagem de input – quando mais profunda a rede, maior será o número total dessas duas operações aplicadas à imagem de input. Esta característica permite a adaptação rápida da rede, não só em casos onde é interessante testar uma rede mais profunda, mas também em casos onde a rede é utilizada em problemas onde as imagens de treino tenham tamanho diferente das que foram utilizadas nesta tese. Consequentemente, a profundidade da rede poderá não ser suficiente ou ser em demasia. Adicionalmente, é ainda possível alterar os valores dos filtros de forma mais fácil e eficiente. Todas estas características permitem utilizar diferentes conjuntos de treino cujas imagens de treino não tenham tamanhos iguais sem ser necessário aplicar outros passos de pré-processamento às imagens.

3.4 Experiências para Afinar a Rede

Antes de realizar os testes para encontrar a melhor máscara de segmentação, foram realizadas experiências para afinar a rede o melhor possível, fazendo variar os seguintes parâmetros: optimizador, taxa de aprendizagem, distância máxima entre píxeis (R), conjunto de filtros, o tamanho do batch e o número de classes (apenas testado com o conjunto de dados das bactérias). Nas experiências realizadas foram utilizados o conjunto de dados dos pokémons e das bactérias. O primeiro conjunto de treino foi utilizado para verificar se a rede não conseguia convergir com algum dos valores selecionado. O segundo conjunto, foi utilizado para selecionar o melhor valor para os diferentes parâmetros.

Relativamente ao número de épocas utilizado no processo de treino, foi implementado o mecanismo de Early Stopping, que consiste em terminar o processo de treino quando a validação do erro de reconstrução não diminui ao fim de N épocas. Neste caso, foi definido que se ao fim de 3 épocas essa validação não diminuísse, o treino terminava. Caso essa situação não ocorra ao fim de 200 épocas, o treino também é dado por terminado. Este mecanismo só não foi utilizado nas experiências para a obtenção do melhor optimizador e taxa de aprendizagem. Nesta situação, foi decidido que se treinava usando 50 épocas porque existia a possibilidade de a rede demorar muito tempo a convergir, o que levava a perder mais tempo com parâmetros que não seriam utilizados.

Nas secções seguintes serão explicadas as experiências realizadas para afinar a rede o melhor possível como os valores selecionados que otimizaram a rede.

3.4.1 Tamanho do Batch

O tamanho do batch relaciona-se com o número de exemplos do conjunto de dados utilizado em cada iteração do ciclo de treino. Quanto menor for este valor, maior o número de exemplos e, consequentemente, maior o tempo de cada iteração. Foram realizadas diversas experiências onde se alterou o tamanho do batch de forma a visualizar os efeitos dessas mesmas alterações nos resultados da rede. No entanto, os resultados obtidos foram bastante semelhantes no que toca às imagens de reconstrução bem como às imagens segmentadas. A diferença principal encontrava-se na duração do treino da rede, ou seja, com um valor de batch maior, a rede demora menos tempo a treinar, como era expectável. Foram realizados testes com os seguintes tamanhos de batchs: 2, 4, 8 e 16, tendo sido selecionado o valor de 8 para este parâmetro da rede. Não foi utilizado um valor maior devido a limitações de hardware, mas é aconselhado, sempre que possível, aumentar este valor.

3.4.2 Optimizador e Taxa de Aprendizagem

Os optimizadores são métodos utilizados para alterar os atributos da rede neural, como os pesos (parâmetros da rede aprendidos durante o processo de treino), de forma a reduzir a função de perda, podendo ajudar a obter resultados mais rápido.

Por outro lado, a taxa de aprendizagem é um hiperparâmetro do optimizador que controla a rapidez com que a rede aprende. Assim, este é um valor que se define antes do treino e que não é alterado durante o processo de treino. Como tal, o optimizador e a taxa de aprendizagem foram testados juntos uma vez que a segunda é um hiperparâmetro do primeiro.

Esta fase de experiências teve como objetivo encontrar o optimizador e a taxa de aprendizagem que melhor se adaptavam aos dados em questão. Foram selecionados dois optimizadores distintos para comparação: Adam e SGD com momentum. A razão principal da escolha do primeiro centra-se na sua rapidez. No entanto, o Adam pode apresentar alguns problemas de convergência em situações em que, normalmente, o SGD com momentum consegue convergir melhor, precisando apenas de mais de tempo de treino. Assim, este foi o segundo optimizador escolhido. O momentum tem como objetivo principal acelerar o processo de treino do modelo e reduzir as oscilações, sendo que foram escolhidos os seguintes dois valores para este hiperparâmetro: 0.5 e 0.9. Apesar de ser possível alterar também o momentum do optimizador Adam, foi escolhido não o fazer. Por último, os valores selecionados para a taxa de aprendizagem foram os seguintes: 0.005, 0.003, 0.001, 0.05, 0.03, 0.01, 0.1, 0.3, 0.5, 0.9.

Nestas experiências, ao invés de se utilizar o Early Stopping referido anteriormente, definiu-se o número de épocas igual a 50. Isto porque ao utilizarmos o Early Stopping a rede poderia não chegar a convergir pois o optimizador ou a taxa de aprendizagem não eram o mais apropriado. Assim, garantiu-se que não se demorava muito tempo a tentar treinar uma rede que não convergia. Os restantes parâmetros mantiveram-se constantes

neste conjunto de experiências: o número de classes foi igual a 5, a distância entre os píxeis igual a 2, os filtros foram os seguintes 16, 32, 64, 128, 256 e o tamanho do batch utilizado foi igual a 8.

Na Tabela 3.1, encontram-se os valores obtidos da função de perda de reconstrução obtidos após 50 épocas com os diferentes conjuntos de dados, optimizadores, taxas de aprendizagem e momentum selecionados. Adicionalmente, na Figura 3.11, encontra-se a evolução da função de perda de reconstrução durante o processo de treino utilizando o conjunto de dados dos pokémons, para ambos os optimizadores. Na Figura 3.12 é apresentado a evolução da função de perda de reconstrução com o conjunto de dados das bactérias, utilizando o optimizador Adam e as diferentes taxas de aprendizagem selecionadas. Em ambos os gráficos, as linhas em tons de laranja representam o conjunto de dados das bactérias e as linhas em tons de azul representam as imagens dos pokémons.

SGD		Adam	Pokemons	All_Channels
Taxa de Aprendizagem	Momentum	Taxa de aprendizagem	Função de Perda de Reconstrução	Função de Perda de Reconstrução
0.01	0.9	0.01	1.032	
0.1	0.9	0.1	58.82	
0.5	0.9	0.5	33.85	
0.9	0.9	0.9	49.57	
0.1	0.5	0.1	23.8	
0.5	0.5	0.5	52.65	
0.9	0.5	0.9	55.29	
0.005		0.005	0.4385	1.473
0.003		0.003	0.2833	0.4773
0.001		0.001	0.1064	0.4186
0.05		0.05	2.263	0.703
0.03		0.03	0.9255	
0.01		0.01	0.8132	0.6707
0.1		0.1	4.054	
0.3		0.3	53.54	

Tabela 3.1: Resultados - Optimizador e Taxa de Aprendizagem

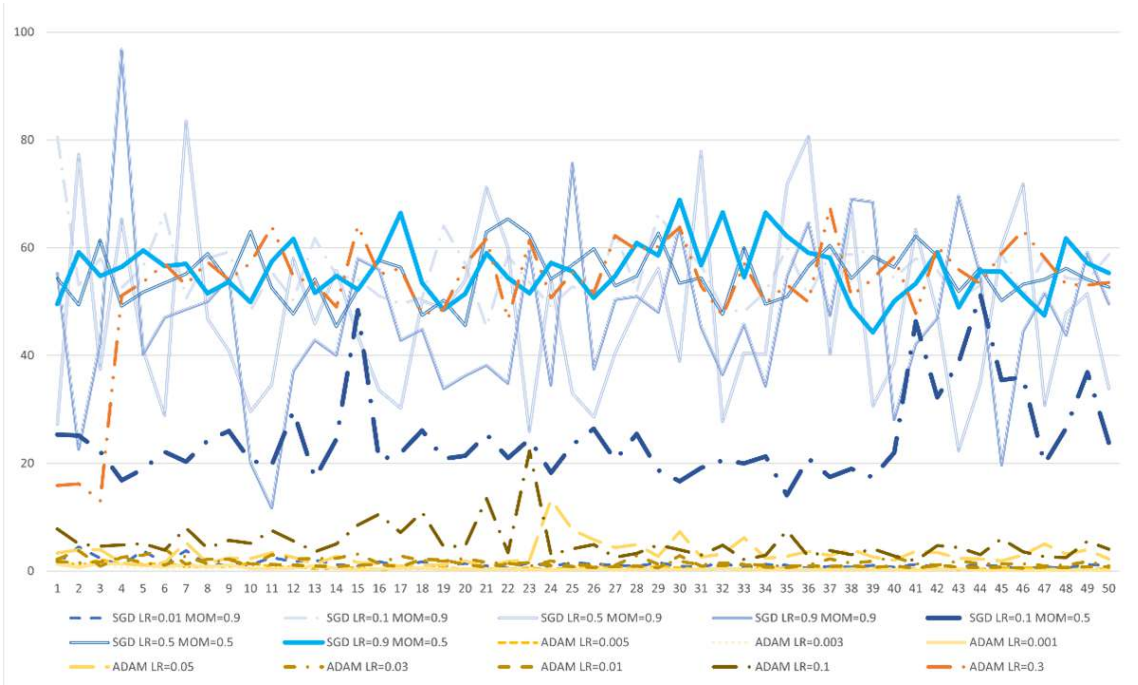


Figura 3.11: Comparação Optimizadores Adam e SGD com Momentum

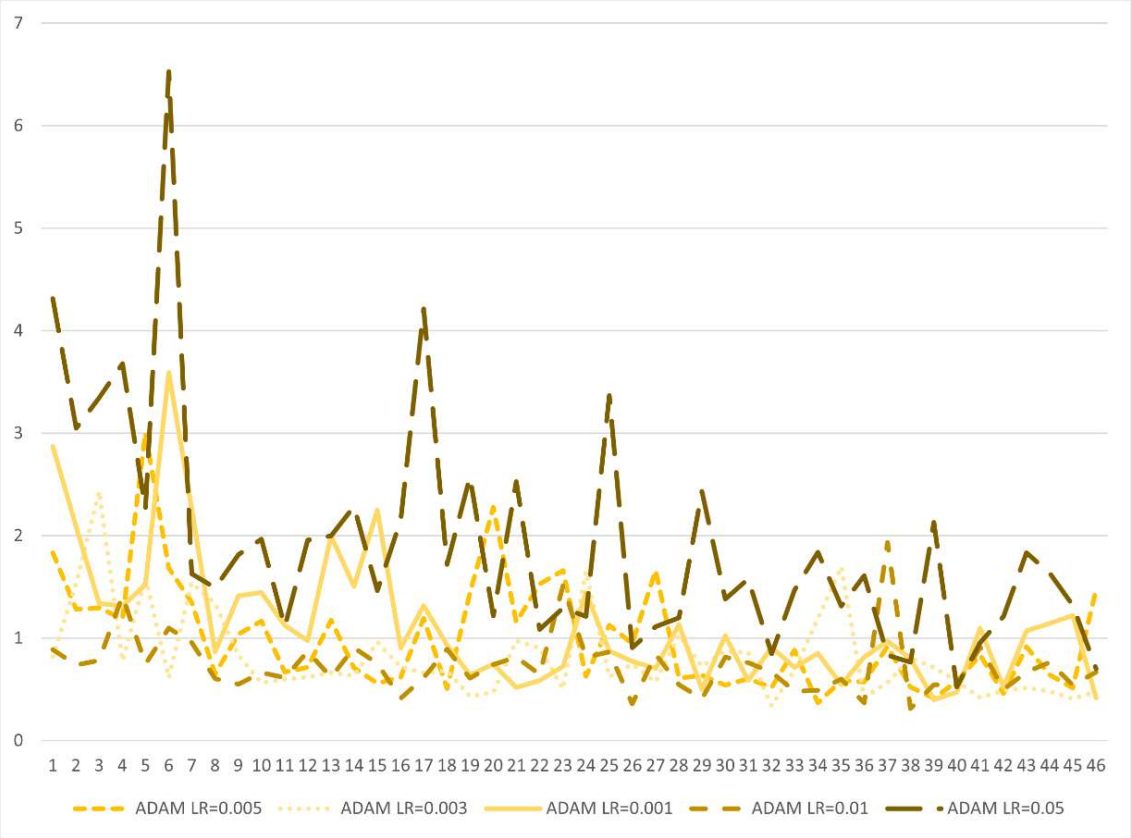


Figura 3.12: Comparação Taxas de Aprendizagens

Pela observação dos Gráficos 3.11, 3.12 e da Tabela 3.1, é possível concluir que, ao fim de 50 épocas a rede, com o optimizador SGD com momentum não aprendeu praticamente nada pois o a função de perda de reconstrução diminuiu ligeiramente ao longo das épocas. Pelo contrário, com o optimizador Adam, a rede aprendeu um pouco uma vez que a função de perda de reconstrução diminuiu. Para além disso, quando se comparam os valores da função de perda de reconstrução obtidos com o SGD e com o Adam, verifica-se que os valores obtidos para o primeiro optimizador são bastante mais elevados quando comparados com o segundo. Assim, conclui-se que para o caso particular desta tese, o optimizador Adam se adapta melhor aos dados em questão, tendo sido o selecionado.

Ao comparar os resultados presentes no gráfico 3.12 e na Tabela 3.1, relativamente à taxa de aprendizagem, chega-se à conclusão de que para uma taxa de aprendizagem igual a 0.001, é obtido o valor mais baixo entre todas as taxas selecionadas. No entanto, para a taxa de aprendizagem igual a 0.01, o valor da perda de reconstrução obtido na última iteração não é consideravelmente mais elevado e obteve menos oscilações, pelo que foi selecionado para a melhor taxa de aprendizagem.

3.4.3 Distância Máxima entre Píxeis Considerada (R)

A distância máxima entre píxeis é utilizada no cálculo dos pesos da função de perda corte normalizado suave. Este hiperparâmetro ajuda na deteção dos limites dos objetos

presentes nas imagens e indica a distância entre cada dois píxeis que deve ser considerada para efeitos do cálculo dos pesos. A distância máxima entre píxeis considerada pode influenciar mais o resultado obtido com conjunto de dados das bactérias do que com o conjunto de dados dos pokémons. O primeiro conjunto é composto por imagens de microscopia o que implica que a passagem entre os objetos e o fundo é mais ténue do que no caso do segundo conjunto mencionado. Adicionalmente, as imagens dos pokémons contêm uma linha delimitadora, o que ajuda na deteção dos limites dos objetos. Deste modo, é bastante importante escolher um bom valor para a distância máxima entre os píxeis considerada. Se for demasiado baixo, há a possibilidade de se estar a desprezar parte da bactéria. Se for demasiado elevado, existe a possibilidade de se incluir numa dada bactéria, parte de outra que está encostada à primeira bactéria. Assim, foram testados os seguintes valores para a distância máxima entre cada dois pixéis: 1, 2, 3, 4 e 5, utilizando apenas o conjunto de dados das bactérias.

Os outros parâmetros mantiveram-se constantes, assumindo os seguintes valores: o optimizador foi o Adam, a taxa de aprendizagem foi igual a 0.003, o tamanho do batch foi igual a 8, o número de classes foi igual a 3 e os filtros utilizados foram: 32, 64, 128 e 256.

Na Tabela 3.2 são apresentados os valores da perda de reconstrução obtidos nas diferentes experiências efetuadas, utilizando o conjunto de dados das bactérias.

R	Função de Perda de Reconstrução
1	1.904
2	1.231
3	0.9514
4	1.323
5	0.7785

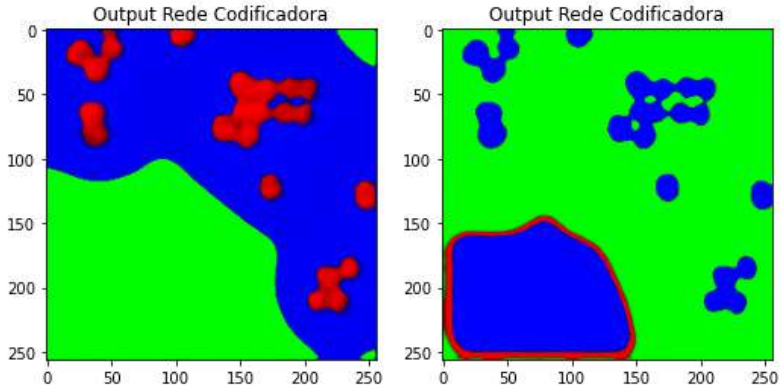
Tabela 3.2: Resultados - Distância Máxima entre Píxeis (R)

Ao comparar os valores obtidos da função de perda de reconstrução presentes na Tabela 3.2 verifica-se que a rede obtém melhores resultados quando o R é igual a 3 e a 5, onde ambos são menores que 1. No entanto, é com R=5 que a rede obtém um valor da função de perda de reconstrução mais baixo. Na Figura 3.13 são apresentados os outputs da rede codificadora obtidos com ambos os valores.

Ao comparar o output da rede codificadora obtido com R igual a 3 e a 5, observa-se que no segundo caso as bactérias ficam ligeiramente mais bem definidas. Como também se obteve um valor da função de perda de reconstrução mais baixo com uma distância máxima entre píxeis igual a 5, decidiu-se que este era o valor apropriado para este hiperparâmetro.

3.4.4 Filtros

O valor dos filtros utlizados nas convoluções de cada módulo da rede é fornecido como um vetor de números inteiros que são utilizados para dois fins distintos. Primeiro, cada elemento definido será utilizado como valor dos filtros em cada módulo da W-Net. Por



(a) R = 3 (b) R = 5

Figura 3.13: Comparação R=3 e R=5, com classe1=Vermelho, classe2=verde, classe3=azul

outro lado, o número total de filtros indica a profundidade máxima que a rede codificadora e a descodificadora assumirão. Por exemplo, se forem escolhidos três filtros, ambas as redes terão uma profundidade igual a 3. Assim, ao selecionar o conjunto de filtros a utilizar é fundamental ter em conta a profundidade máxima da rede. Isto porque uma rede mais profunda não significa, necessariamente, que obterá melhores resultados. Adicionalmente, deve-se evitar problemas como Vanishing Gradient ou o aumento exponencial do modelo com o aumento da profundidade.

Deste modo, para evitar esses problemas, foram realizadas diversas experiências com o objetivo de encontrar o melhor conjunto de filtros bem como o total de filtros mais indicados para os conjuntos de dados das bactérias. Assim, treinou-se o modelo com os seguintes conjuntos de filtros: [16, 32, 64], [32, 64, 128], [64, 128, 256], [128, 256, 512], [16, 32, 64, 128], [32, 64, 128, 256], [64, 128, 256, 512], [128, 256, 512, 1024], [16, 32, 64, 128, 256], [32, 64, 128, 256, 512], [64, 128, 256, 512, 1024]. Os restantes parâmetros mantiveram-se constantes com os seguintes valores: o optimizador foi o Adam, a taxa de aprendizagem igual a 0.003, o tamanho do batch igual a 4, o número de classes foi igual a 3 e, por último, a distância entre os pixéis igual a 2.

A Tabela 3.3 apresenta o valor da função de perda de reconstrução, a validação da função de perda de reconstrução, a função de perda do corte normalizado suave e a validação da função de perda do corte normalizado suave obtidos para o conjunto de dados dos pokémons.

Ao longo de todas as experiências efetuadas para a obtenção do melhor conjunto de filtros, o valor da perda do corte normalizado suave manteve-se relativamente baixo e não variou significativamente entre os diferentes testes. Assim, a decisão do melhor conjunto de filtros centrou-se na evolução da perda de reconstrução bem como no output da rede codificadora e nas máscaras de segmentação obtidas.

Ao treinar a rede com as imagens dos pokémons, os seguintes conjuntos de filtros obtiveram um valor da perda de reconstrução mais baixo: [32, 64, 128] e [64, 128, 256, 512, 1024]. Mas, como a diferença desta função de perda entre os diferentes conjuntos não

Conjuntos de Filtros	Conjunto de dados pokémons			
	Perda $J_{Soft-NCut}$	Validação da Perda $J_{Soft-NCut}$	Perda de Reconstrução	Validação Perda de Reconstrução
[16, 32, 64]	2.131	2.161	0.7326	0.7326
[32, 64, 128]	2.011	2.014	0.2758	0.2758
[64, 128, 256]	2.117	2.129	0.5359	0.5359
[128, 256, 512]	2.077	2.072	0.6789	0.6789
[16, 32, 64, 128]	2.223	2.27	1.052	1.052
[32, 64, 128, 256]	2.163	2.141	0.5523	0.5523
[64, 128, 256, 512]	2.121	2.073	0.6154	0.6154
[128, 256, 512, 1024]	2.042	2.069	0.3621	0.361
[16, 32, 64, 128, 256]	2.023	2.028	0.4317	0.3621
[32, 64, 128, 256, 512]	2.041	2.031	0.5534	0.5534
[64, 128, 256, 512, 1024]	2.001	2.004	0.2669	0.2669

Tabela 3.3: Resultados - Filtros Imagens Pokémons

foi muito elevada, utilizaram-se os mesmos conjuntos de filtros para treinar a rede com as imagens das bactérias. Assim, na Tabela 3.4 é apresentado o valor obtido da função de perda de reconstrução, a validação da perda de reconstrução, a função de perda do corte normalizado suave e a validação da função de perda do corte normalizado suave obtidos para o conjunto de dados das bactérias.

Conjuntos de Filtros	Conjunto de dados bactérias			
	Perda $J_{Soft-NCut}$	Validação Perda $J_{Soft-NCut}$	Perda de Reconstrução	Validação Perda de Reconstrução
[16, 32, 64]	2.101	2.101	1.499	1.499
[32, 64, 128]	2.064	2.043	0.769	0.769
[64, 128, 256]	2.249	2.012	3.362	3.362
[128, 256, 512]	2.051	2.027	1.519	1.519
[16, 32, 64, 128]	2.049	2.02	1.669	1.669
[32, 64, 128, 256]	2.06	2.057	0.8024	0.8024
[64, 128, 256, 512]	2.015	2.055	1.166	1.166
[128, 256, 512, 1024]	2.012	2.024	0.7421	0.7421
[16, 32, 64, 128, 256]	2.007	2.011	0.7918	0.7918
[32, 64, 128, 256, 512]	2.025	2.019	0.7365	0.7365
[64, 128, 256, 512, 1024]	2.045	2.167	0.6113	0.6113

Tabela 3.4: Resultados - Filtros Imagens Bactérias

Ao realizar as mesmas experiências com o conjunto de dados das bactérias, foi possível verificar que existiram vários conjuntos de filtros que obtiveram valores da perda de reconstrução baixos e semelhantes. Entre eles encontram-se os seguintes: [32, 64, 128], [128, 256, 512, 1024], [16, 32, 64, 128, 256], [32, 64, 128, 256, 512] e [64, 128, 256, 512, 1024]. Assim, para selecionar o melhor conjunto de filtros, comparei também as máscaras de segmentação obtidas com cada um desses conjuntos de filtros. Deste modo, nas Figuras 3.14 a 3.18, são apresentas a imagem inicial, o output da rede codificadora, as melhores máscaras de segmentação obtidas para cada um desses conjuntos de filtros e a imagem de reconstrução.

Ao comparar as máscaras de segmentação obtidas (Figuras 3.14 a 3.18), é possível verificar que as piores correspondem aos conjuntos de filtros com um valor da perda de reconstrução mais baixo, como é o caso do conjunto [64, 128, 256, 512, 1024]. Por outro lado, o conjunto de filtros [16, 32, 64, 128, 256] obteve boas máscaras de segmentação, mas um valor da perda de reconstrução mais alto que o primeiro conjunto mencionado. Apesar disso, como o mais importante é a obtenção de boas máscaras de segmentação, o conjunto de filtros [16, 32, 64, 128, 256] foi o selecionado, apesar de não ter sido o que obteve o menor valor da função de perda de reconstrução.

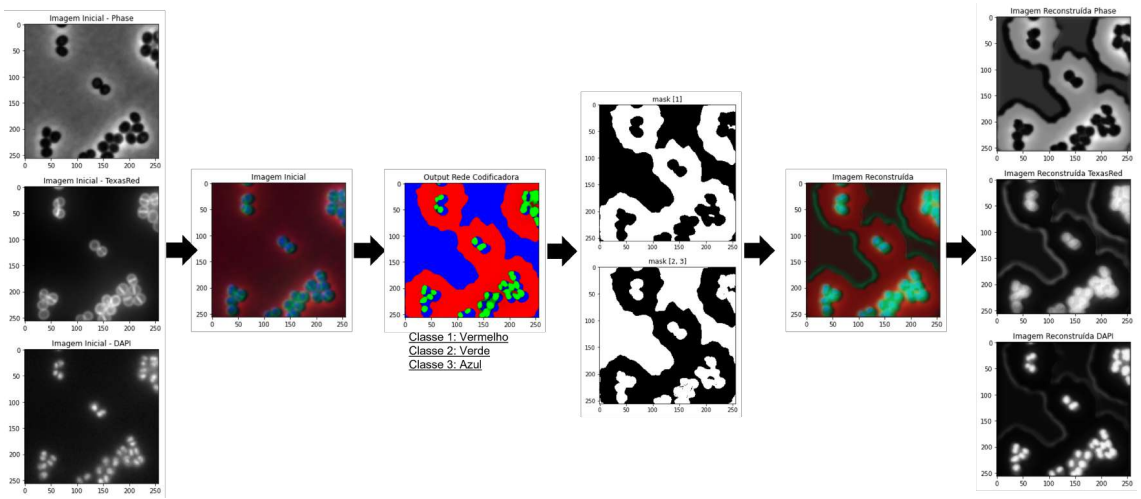


Figura 3.14: Máscaras de Segmentação Obtidas Filtros=32, 64, 128

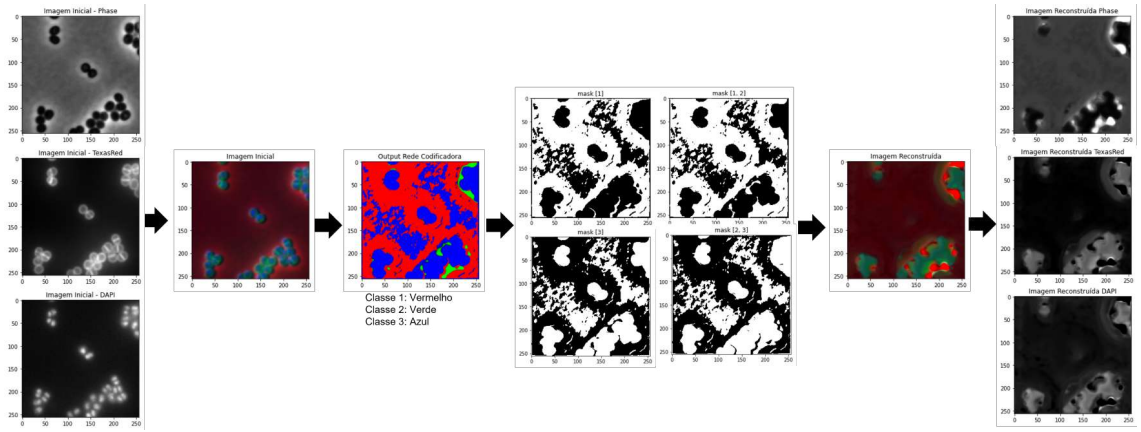


Figura 3.15: Máscaras de Segmentação Obtidas Filtros=128, 256, 512, 1024

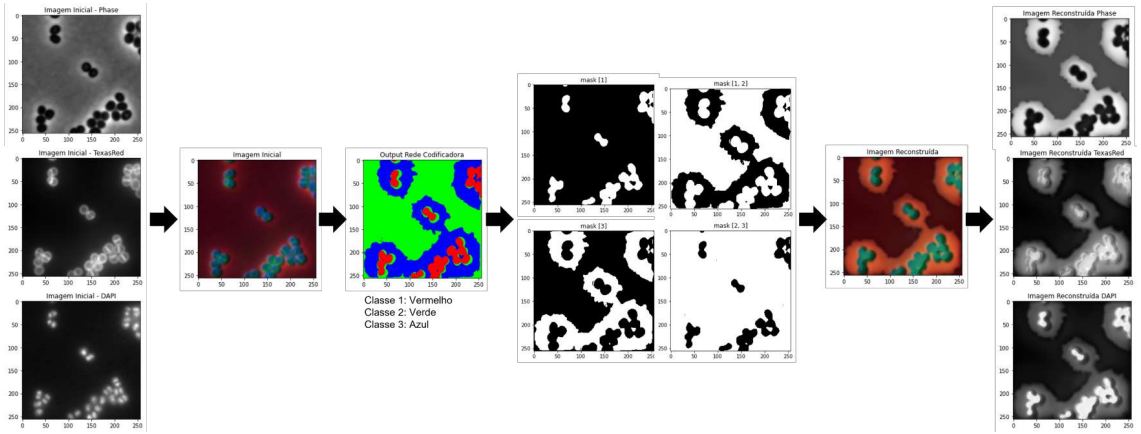


Figura 3.16: Máscaras de Segmentação Obtidas Filtros=16, 32, 64, 128, 256

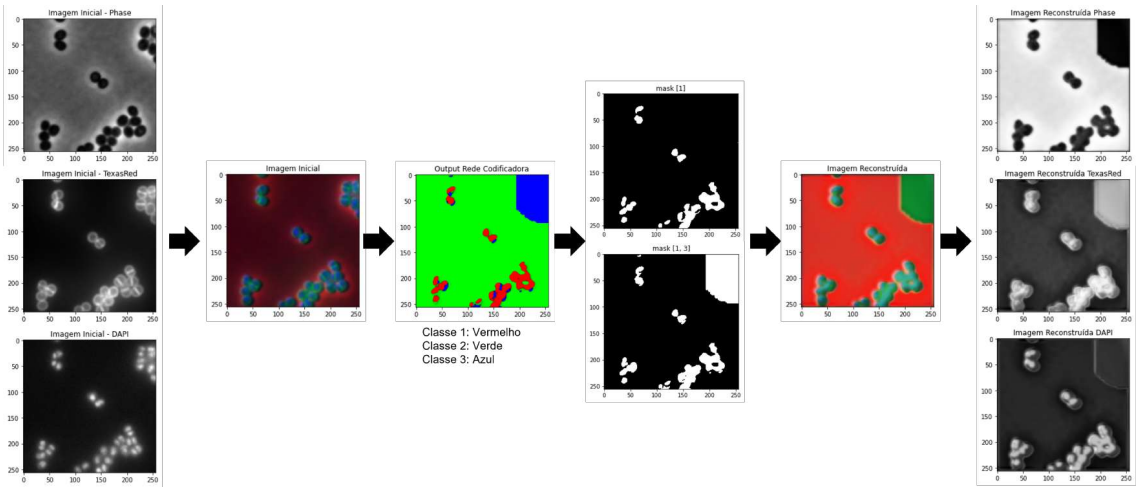


Figura 3.17: Máscaras de Segmentação Obtidas Filtros=32, 64, 128, 256, 512

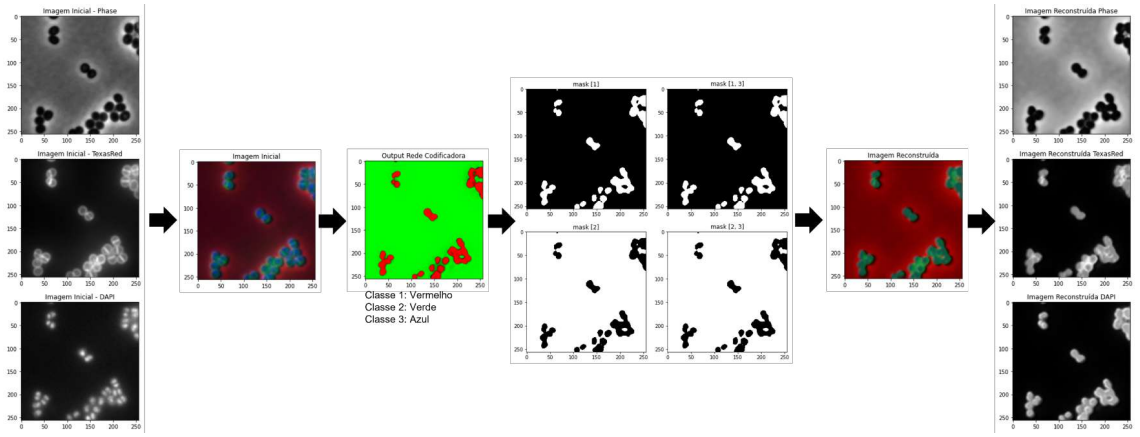


Figura 3.18: Máscaras de Segmentação Obtidas Filtros=64, 128, 256, 512, 1024

4

Resultados e Discussão

Esta dissertação tinha como objetivo resolver dois problemas principais na segmentação de imagens de microscopia de bactérias. O primeiro centrava-se na eliminação da necessidade de utilizar dados rotulados durante o processo de treino e o segundo problema focava-se na correção da segmentação de grupos de bactérias juntas. De forma a tentar resolver ambos os problemas foi implementada a arquitetura W-Net que já tinha mostrado bons resultados na segmentação de imagens de células e utiliza dados sem rótulos no processo de treino.

No capítulo anterior, foram apresentados os conjuntos de experiências e respetivos resultados que levaram à escolha dos seguintes valores dos parâmetros da rede W-Net implementada. Na rede final, foi selecionado o optimizador Adam com uma taxa de aprendizagem igual a 0.01, uma distância máxima entre píxeis igual a 5 e, por último, o conjunto de filtros a utilizar será o seguinte: 16, 32, 64, 128, 256. Estes mesmos valores foram utilizados na rede final e para encontrar a melhor máscara de segmentação. Para isso, a rede foi treinada com as imagens das bactérias e com os seguintes números de classes: 2, 3, 4 e 5. Para cada um desses valores, foram obtidas boas máscaras de segmentação que são apresentadas nas Figuras 4.1 a 4.4.

Começando com o número de classes igual a 2 (Figura 4.1), as melhores máscaras de segmentação correspondem às obtidas com as classes 1 e 2 separadas. Apesar de se ter obtido boas máscaras de segmentação, estas não apresentam qualquer divisão entre as bactérias. Adicionalmente, as imagens reconstruídas ficaram bastante piores quando comparadas com as imagens reconstruídas dos restantes números de classes.

Quando se treinou a rede com um número de classes igual a 3 (Figura 4.2), conseguiu-se obter boas segmentações com as seguintes combinações de classes: 1, 2, 1+3 e 2+3. No entanto, as máscaras obtidas com as duas últimas ficaram iguais às obtidas com a classe 1 e a classe 2, respetivamente. Consequentemente, podemos concluir que a classe 3 é redundante, não fornecendo nenhuma informação adicional às máscaras de segmentação obtidas com a classe 1 e 2. Ao contrário da imagem reconstruída obtida com duas classes, com 3 classes, foi obtida uma imagem reconstruída semelhante à imagem inicial.

Em relação ao número de classes igual a 4 (Figura 4.3), foi obtida uma boa máscara de

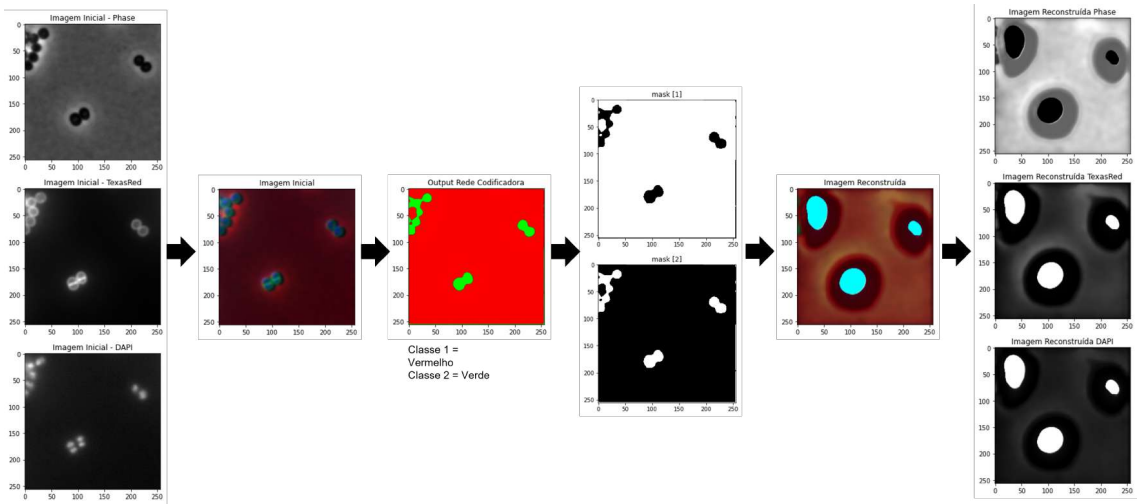


Figura 4.1: Máscaras de Segmentação Obtidas Número de Classes = 2 Bactérias

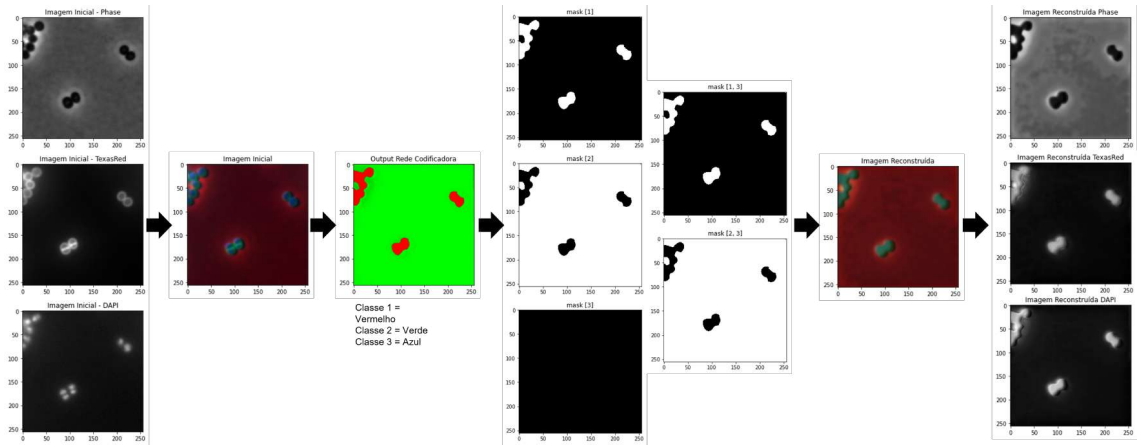


Figura 4.2: Máscaras de Segmentação Obtidas Número de Classes = 3 Bactérias

segmentação com a classe 1. Com a classe 2, apesar da segmentação obtida ser promissora, esta apresenta alguns erros, não sendo tão precisa em certas células quando comparadas com a imagem inicial. As combinações de classes 1+3, 1+4, 1+3+4, 2+3, 2+4 e 2+3+4 obtiveram boas segmentações. Contudo, as máscaras obtidas com as três primeiras são iguais à máscara obtida com a classe 1 e as máscaras obtidas com as últimas três são iguais à segmentação obtida com a classe 2. Assim, pode-se concluir que as classes 3 e 4 não contém qualquer informação relevante na geração de máscaras de segmentação, sendo ambos redundantes.

Por último, com o número de classes igual a 5 (Figura 4.4), obteve-se uma boa máscara de segmentação com a classe 1. Nas máscaras obtidas com as classes 2 e 3 as bactérias foram identificadas, mas os grupos maiores de bactérias ficaram mal segmentados. Nestes casos, as bactérias foram segmentadas como uma célula grande, não existindo qualquer identificação das células que se encontram no meio do grupo. Por outro lado, as segmentações obtidas com as classes 1+4, 1+5 e 1+4+5 são bastante boas, mas semelhantes à máscara obtida com a classe 1. Conclui-se assim, que as classes 4 e 5 não acrescentam

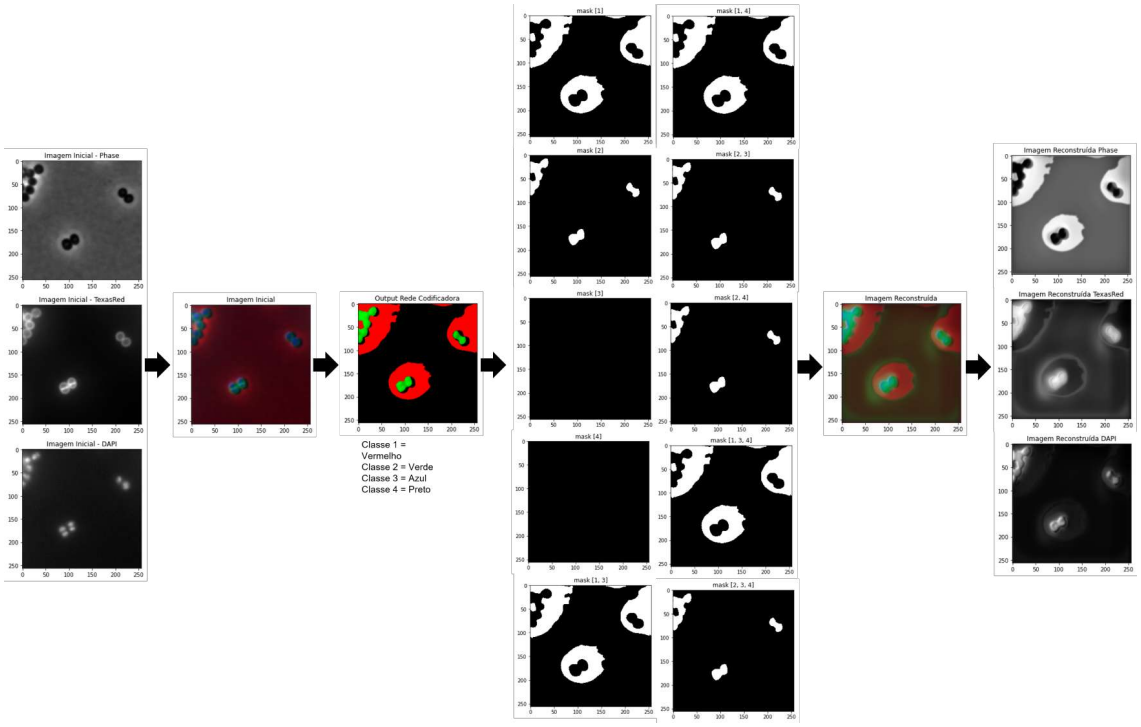


Figura 4.3: Máscaras de Segmentação Obtidas Número de Classes = 4 Bactérias

nenhuma informação útil na criação de máscaras de segmentação, sendo ambos redundantes.

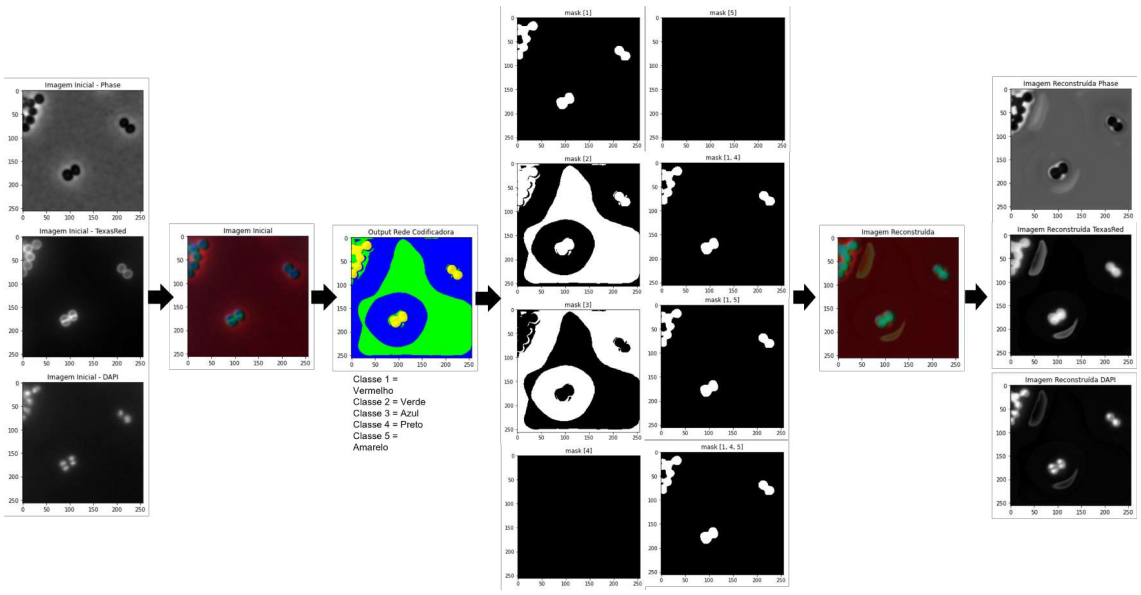


Figura 4.4: Máscaras de Segmentação Obtidas Número de Classes = 5 Bactérias

As máscaras de segmentação obtidas mostraram que é possível segmentar imagens de microscopia de bactérias sem a utilização de dados rotulados ao utilizar a arquitetura W-Net. Todavia, existem dois problemas principais que devem ser endereçados: a separação das bactérias e o delineamento das mesmas. Como o segundo objetivo deste trabalho era resolver o problema da correta segmentação dos grupos de bactérias juntas, é proposto

o seguinte passo de pós-processamento para esse efeito. Este consiste na aplicação do algoritmo de segmentação Watershed que é bastante utilizado para separar diferentes objetos presentes numa imagem. Neste algoritmo [35] a imagem é vista como sendo um perfil topográfico, onde os valores de cada elemento da imagem ou intensidade representam a altitude desse ponto. A analogia deste tipo de segmentação é baseada na ideia de que uma gota de água virtual fluiria para o mínimo de intensidade local da imagem. Nos pontos onde a gota flui para mais do que um mínimo, existe uma bacia hidrográfica, que divide as nascentes adjacentes e, consequentemente, os objetos adjacentes. Assim, o relevo consiste em vales baixos (mínimos), cristas de grande altitude (linhas das bacias hidrográficas) e encostas (bacias de captação). O objetivo principal deste método de segmentação é determinar a localização de todas as bacias de captação e/ou as linhas da bacia hidrográfica pois estas são consideradas um segmento separador da imagem. Na Figura 4.5, encontra-se um exemplo da aplicação do algoritmo Watershed implementado na biblioteca Skimage. O algoritmo é aplicado à máscara de segmentação obtida pelo processo anteriormente explicado. No caso da Figura 4.5, foi aplicado à segmentação obtida com 3 classes, utilizando apenas a primeira classe.

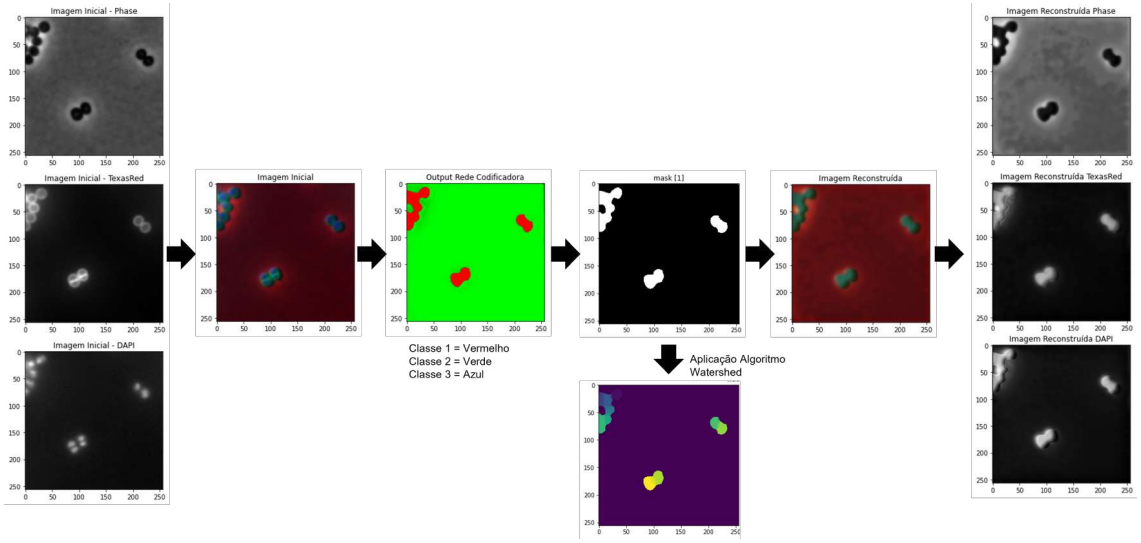


Figura 4.5: Exemplo Aplicação Segmentação Watershed

A Figura 4.5 mostra que aplicar o algoritmo Watershed à máscara de segmentação resultante da combinação de classes como passo de pós-processamento, consegue-se obter uma separação quase perfeita das bactérias que se encontram juntas. Todavia, o primeiro problema mencionado, o do correto delineamento das bactérias, deverá ser endereçado no futuro. Devido ao pouco delineamento das bactérias, algumas das que se encontram presentes nos grupos com um maior número de células não se encontravam devidamente delimitadas, tendo uma transição suave de cores entre as mesmas. O conjunto Vermelho Texas caracteriza-se por evidenciar o delineamento dos objetos e o conjunto DAPI identifica o interior das bactérias. Se conseguíssemos combinar as informações destas duas classes, poderia ser possível obter um melhor delineamento das bactérias. No entanto, ao

observar as imagens reconstruídas presentes nas Figurass 4.1 a 4.4, verifica-se que estes dois conjuntos ficaram bastante aquém da imagem inicial correspondente, indicando que a rede ainda tem espaço de melhorias. Uma das possíveis melhorias seria prolongar o tempo de treino de forma que a rede consiga aprender os detalhes das imagens e não se fique apenas pelo grande plano. Outras soluções centravam-se em encontrar outros valores para os parâmetros da rede que conseguissem melhorar essas imagens. No entanto, devido a restrições de tempo não foi possível terminar essas experiências.

Concluindo, a arquitetura W-net pode ser utilizada para segmentação de imagens de microscopia de bactérias. Para isso, criam-se as máscaras de segmentação a partir das combinações das classes do output da rede codificadora. Para resolver o problema associado aos grupos de bactérias, é proposto a aplicação do algoritmo Watershed que tem como característica principal dividir os objetos que se encontram juntos. No entanto, é necessário corrigir alguns problemas de delineamento dos objetos, principalmente em situações em que existem grupos com mais do que duas bactérias. Para isso, sugere-se a melhoria dos resultados obtidos com os canais Vermelho Texas e DAPI e, de seguida, combinar as informações destes dois canais para melhoramento do delineamento.

5

Conclusões e Trabalho Futuro

A utilização de conjuntos de dados já rotulados requerem a seleção manual de cada objeto presente na imagem. No caso do conjunto de dados utilizado nesta dissertação esta rotulação implicava selecionar todas as bactérias que se encontravam em todas as imagens utilizadas, o que é uma tarefa bastante trabalhosa e demorada. Por outro lado, trabalhos anteriores de segmentação de bactérias apresentavam problemas na segmentação de grupos de bactérias ao segmentá-las como uma só. Assim, o objetivo deste trabalho foi encontrar uma forma de resolver ambos os problemas mencionados.

De forma a realizar a segmentação de imagens de microscopia de bactérias, foi implementada uma rede baseada na arquitetura W-Net. Esta é composta por duas redes, a codificadora e a descodificadora, que juntas são vistas como um autoencoder. Este é uma rede neural que utiliza dados não rotulados durante o processo de treino. Ao utilizar a arquitetura W-Net resolveu-se o primeiro problema mencionado ao utilizar um conjunto de dados sem qualquer tipo de identificação das diferentes bactérias de cada imagem.

O output da rede codificadora tem a seguinte dimensão 256x256xK, onde o K representa o número de classes. Este output é o resultado da aplicação da ativação softmax que é uma distribuição de probabilidades que indica a probabilidade, para cada píxel da imagem, de este pertencer a cada uma das classes previamente definidas. A soma de todas essas probabilidades ou ativações de todos as classes, para cada píxel, encontra-se compreendida entre 0 e 1.

Utilizando o número de classes definido antecipadamente, foram calculadas as diferentes combinações possíveis de classes. Para cada combinação de classes existente, realiza-se a soma das ativações das classes respetivas utilizando, para isso, o output da rede codificadora. Caso essa soma seja inferior a 0.5, a máscara de segmentação é 0 nesse ponto. Caso seja maior ou igual a 0.5, a máscara é igual a 1 nesse ponto.

Por exemplo, ao utilizar um número de classes igual a 3, obtém-se as seguintes combinações: 1, 2, 3, 1+2, 1+3, 2+3, 1+2+3. Para criar a máscara de segmentação resultante da combinação 1+2, começa-se por realizar a soma dessas duas classes. De seguida, para cada ponto, verifica-se se o valor é menor que 0.5. Se for, passa a 0, senão passa a 1. No caso das combinações com apenas uma classe, não se efetuam somas. Assim, a obtenção da

máscara de segmentação passa apenas pelo segundo passo. Na Figura 5.1 são apresentas as máscaras de segmentação obtidas com as classes 1 e 2.

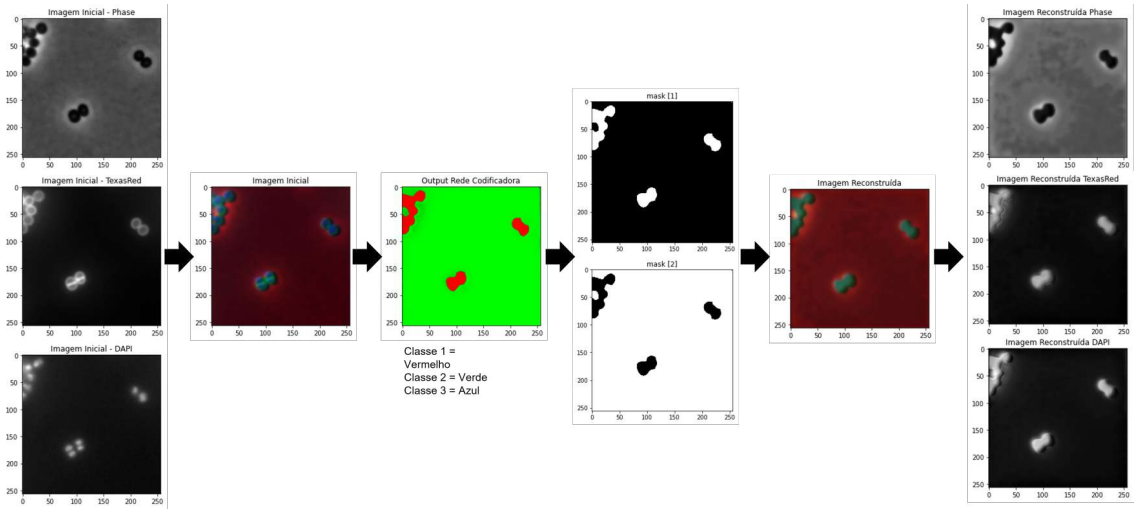


Figura 5.1: Exemplo da Criação Máscaras de Segmentação (Número de Classes = 3)

As máscaras de segmentação obtidas com as classes 1 e 2 foram as melhores máscaras de segmentação obtidas com o número de classes igual a 3. Para além do número de classes igual a 3, houve outros que obtiveram boas máscaras de segmentação com certas combinações de classes. Foi o caso do número de classes igual a 2, 4 e 5. No primeiro caso, as melhores máscaras obtiveram-se com as combinações 1, 2. Com o número de classes igual a 4, obtiveram-se boas máscaras de segmentação com as combinações 1, 2. No último caso, as combinações de classes que originaram as melhores máscaras de segmentação foram 1, 2 e 3.

Apesar dos resultados razoáveis obtidos, estes apresentam alguns problemas quer nas imagens recriadas quer nas máscaras de segmentação. Começando pelas imagens reconstruídas dos diferentes canais (Phase, Vermelho Texas e DAPI), observa-se que a recriação do canal Phase ficou bastante semelhante ao original. No entanto, o mesmo já não acontece com os restantes dois canais, onde a rede foi incapaz de replicar os detalhes dos canais respetivos. No caso do canal Vermelho Texas, ao invés de se conseguir o limite aproximado das bactérias, a imagem recriada ficou um pouco esbatida. No caso do canal DAPI, obteve-se as bactérias juntas em vez do interior das mesmas. Uma forma de corrigir este problema seria deixar a rede treinar durante mais tempo visto que esta consegue distinguir as bactérias do fundo, mas já não distingue os detalhes das bactérias. No entanto, devido a restrições de tempo, não foi possível treinar a rede durante mais tempo para melhorar estes resultados.

Relativamente às máscaras de segmentação obtidas, estas ainda apresentam dois problemas principais. O primeiro centra-se no delineamento das bactérias que, em casos de várias bactérias juntas, encontra-se um pouco deficiente. Por outro lado, as bactérias

não se encontram segmentadas separadamente nas máscaras de segmentação apresentadas. Para resolver o segundo problema, foi proposto a utilização de um passo de pós-processamento para efetuar a separação das bactérias e obter a máscara de segmentação final. Foi aplicado o algoritmo de segmentação Watershed que é bastante utilizado na separação de objetos juntos presentes numa imagem. Neste algoritmo, a imagem é vista como sendo um perfil topográfico, onde os valores de cada elemento da imagem ou intensidade representam a altitude desse ponto. A analogia deste tipo de segmentação é baseada na ideia de que uma gota de água virtual fluiria para o mínimo de intensidade local da imagem. Nos pontos onde a gota flui para mais do que um mínimo, existe uma bacia hidrográfica, que divide as nascentes adjacentes e, consequentemente, os objetos adjacentes. Um exemplo da aplicação deste algoritmo encontra-se na Figura 5.2, onde se aplicou a segmentação Watershed implementada na biblioteca Skimage à máscara de segmentação obtida com a classe 1 e com o número de classes igual a 3.

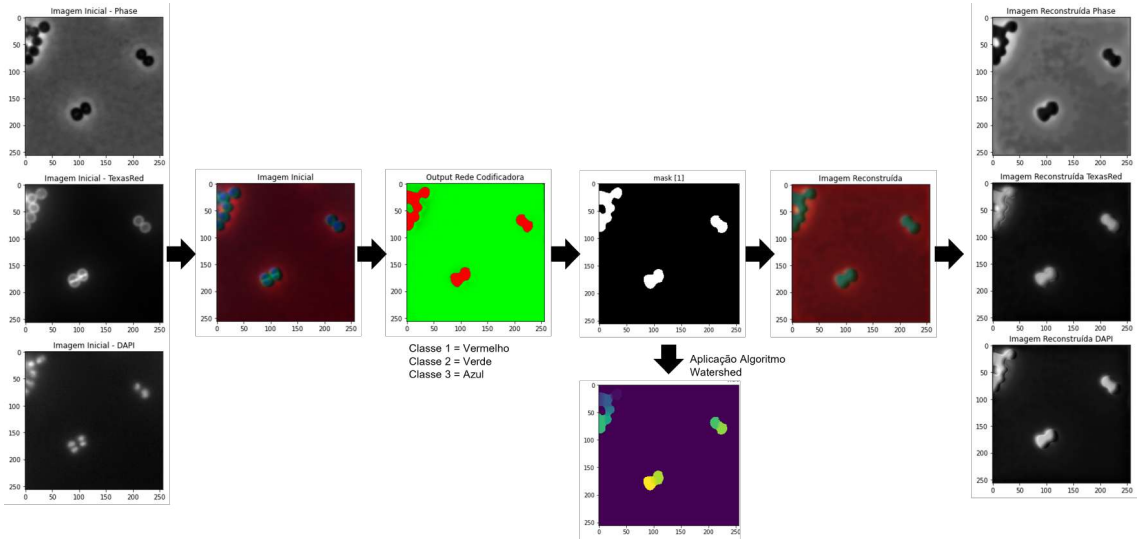


Figura 5.2: Exemplo Aplicação Segmentação Watershed

A Figura 5.2 mostra que ao aplicar o algoritmo Watershed como passo de pós processamento à máscara de segmentação, obtém-se uma separação entre as diferentes bactérias. No entanto, como as máscaras de segmentação obtidas ainda não possuem os limites das bactérias totalmente definidos, ainda existem situações em que há uma mudança de cor gradual entre as bactérias ao invés de uma repentina, indicando o fim de uma célula e o começo de outra.

Assim, o trabalho a efetuar no futuro centra-se em corrigir o problema do delineamento das bactérias. Para isso, sugere-se corrigir o problema das imagens de reconstrução dos canais Vermelho Texas e DAPI de forma a garantir que a rede está a aprender não só as informações gerais, mas, também, os detalhes das imagens. De seguida, como o canal Vermelho Texas se foca no limite das bactérias e o DAPI se foca no interior das mesmas, sugere-se a combinação destas duas informações de forma a resolver o problema do delineamento das bactérias.

Resumidamente, com esta dissertação foi possível realizar segmentação não supervisionada de imagens de microscopia de bactérias. Para isso, recorreu-se ao output da rede codificadora da arquitetura W-Net e às combinações de classes obtidas com o número de classes para gerar uma máscara de segmentação. Esta era obtida ao somar as classes correspondentes de cada combinação e, para cada ponto, se a soma fosse inferior a 0.5 então esse ponto passaria a 0, caso contrário passaria a 1. De entre todas as máscaras de segmentação, há umas que são consideravelmente melhores que outras, que serão as selecionadas. Todavia, a utilização do processo explicado, não foi suficiente para obter uma máscara de segmentação precisa das imagens de microscopia de bactérias, existindo três problemas principais: a falta de detalhes nas imagens recriadas dos canais Vermelho Texas e DAPI, a separação das bactérias e o seu delineamento. Uma possível solução para o primeiro problema seria deixar a rede a treinar durante mais tempo, o poderia permitir à rede aprender não só os aspetos gerais das imagens iniciais, mas também os detalhes das bactérias. Ao resolver este problema, poder-se-ia utilizar as informações dos canais Vermelho Texas (que se foca do limite das bactérias) e DAPI (que se foca no interior das bactérias) para resolver o problema do delineamento das células. Por último, para resolver o segundo problema foi fundamental a utilização de um algoritmo de pós-processamento para atingir o resultado esperado. Para isso foi proposto a utilização do algoritmo de segmentação Watershed, o que obteve resultados bastante bons.

Para finalizar, no decorrer desta dissertação foi implementado um método de aprendizagem não supervisionada para a segmentação de imagens de microscopia de bactérias. Este processo de segmentação evitou o trabalho de marcar manualmente as diferentes regiões que deveriam ser segmentadas para gerar um conjunto de treino. A evitação da seleção manual das bactérias permitiu poupar o tempo relacionado ao processo de seleção de todas as bactérias de todas as imagens do conjunto de treino, tarefa que é bastante demorada.

Bibliografia

[1] *Activation functions in neural networks*. Out. de 2020. URL: https://www.geeksforgeeks.org/activation-functions-neural-networks/ (ver pp. 34, 35).

[2] P. Arbelaez et al. "Contour Detection and Hierarchical Image Segmentation". Em: *IEEE Trans. Pattern Anal. Mach. Intell.* 33.5 (mai. de 2011), pp. 898–916. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2010.161. URL: https://doi.org/10.1109/TPAMI.2010.161 (ver p. 32).

[3] V. Badrinarayanan, A. Kendall e R. Cipolla. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". Em: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.12 (2017), pp. 2481–2495. DOI: 10.1109/TPAMI.2016.2644615 (ver pp. 23, 24).

[4] N. E. Beckman, K. Bierhoff e J. Aldrich. "Verifying Correct Usage of Atomic Blocks and Typestate". Em: *SIGPLAN Not.* 43.10 (2008), pp. 227–244. ISSN: 0362-1340. DOI: http://doi.acm.org/10.1145/1449955.1449783 (ver p. 4).

[5] J. Brownlee. *Deep Learning With Python: Develop Deep Learning Models on Theano and TensorFlow Using Keras*. Machine Learning Mastery, 2016. URL: https://books.google.pt/books?id=K-ipDwAAQBAJ (ver p. 40).

[6] S. Cai et al. "Dense-UNet: a novel multiphoton in vivo cellular image segmentation model based on a convolutional neural network". Em: *Quantitative Imaging in Medicine and Surgery* 10.6 (2020). ISSN: 2223-4306. URL: https://qims.amegroups.com/article/view/43519 (ver p. 35).

[7] F. Chollet et al. *Keras*. 2015. URL: https://keras.io/api/layers/convolution_layers/separable_convolution2d/ (ver p. 15).

[8] Y. Dawoud et al. "Few-Shot Microscopy Image Cell Segmentation". Em: *arXiv:2007.01671* (2020). eprint: arXiv:2007.01671 (ver p. 11).

[9] J. Duchi, E. Hazan e Y. Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". Em: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.html (ver p. 38).

[10] A. Géron. *Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow*. O'REILLY MEDIA, INC, USA, 2019. ISBN: 9781492032649 (ver p. 7).

[11] S. Ghosh et al. *Understanding Deep Learning Techniques for Image Segmentation*. 2019. arXiv: 1907.06119 [cs.CV] (ver p. 20).

[12] R. Girshick. "Fast R-CNN". Em: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. ICCV '15. USA: IEEE Computer Society, 2015, pp. 1440–1448. ISBN: 9781467383912. DOI: 10.1109/ICCV.2015.169. URL: https://doi.org/10.1109/ICCV.2015.169 (ver p. 19).

[13] R. Girshick et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". Em: (jun. de 2014) (ver p. 19).

[14] I. Goodfellow, Y. Bengio e A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (ver pp. 2, 5, 6, 22).

[15] K. He et al. "Mask R-CNN". Em: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322 (ver p. 20).

[16] D. Hossain et al. "Real-Time Food Intake Monitoring Using Wearable Egocnetric Camera". Em: vol. 2020. Jul. de 2020, pp. 4191–4195. DOI: 10.1109/EMBC44109.2020.9175497 (ver p. 16).

[17] J. D. Hunter. "Matplotlib: A 2D Graphics Environment". Em: *Computing in Science Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55 (ver p. 41).

[18] Y. Ioannou. *A tutorial on filter groups (grouped convolution)*. Ago. de 2017. URL: https://blog.yani.ai/filter-group-tutorial/ (ver p. 17).

[19] J. Jin, A. Dundar e E. Culurciello. *Flattened Convolutional Neural Networks for Feedforward Acceleration*. 2015. arXiv: 1412.5474 [cs.NE] (ver p. 14).

[20] I. Khlestov. *Different types of the convolution layers*. Jul. de 2017. URL: https://ikhlestov.github.io/pages/machine-learning/convolutions-types/ (ver pp. 12, 14, 15).

[21] D. P. Kingma e J. Ba. *Adam: A Method for Stochastic Optimization*. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. 2014. URL: http://arxiv.org/abs/1412.6980 (ver p. 39).

[22] A. Krizhevsky, I. Sutskever e G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". Em: *Advances in Neural Information Processing Systems*. Ed. por F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (ver p. 17).

[23] D. Kucharski et al. "Semi-Supervised Nests of Melanocytes Segmentation Method Using Convolutional Autoencoders". Em: *Sensors* 20.6 (2020). ISSN: 1424-8220. DOI: 10.3390/s20061546. URL: https://www.mdpi.com/1424-8220/20/6/1546 (ver pp. 21, 22, 34).

[24] Y. Li, H. Hong e T. Fang. "Hierarchical Segmentation of Remote Sensing Images by Unsupervised Deep Learning Features". Em: *2017 10th International Symposium on Computational Intelligence and Design (ISCID)* 1 (2017), pp. 448–453 (ver p. 23).

[25] M. Lin, Q. Chen e S. Yan. *Network In Network*. 2014. arXiv: 1312.4400 [cs.NE] (ver p. 13).

[26] Q. Liu et al. *GAN based Unsupervised Segmentation: Should We Match the Exact Number of Objects*. 2020. arXiv: 2010.11438 [cs.CV] (ver p. 42).

[27] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (ver p. ii).

[28] J. Ma. *Segmentation Loss Odyssey*. 2020. arXiv: 2005.13449 [eess.IV] (ver pp. 35, 36).

[29] M. Mahmud, M. S. Kaiser e A. Hussain. *Deep Learning in Mining Biological Data*. 2020. arXiv: 2003.00108 [q-bio.QM] (ver p. 20).

[30] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/ (ver p. 40).

[31] S. Minaee et al. *Image Segmentation Using Deep Learning: A Survey*. 2020. arXiv: 2001.05566 [cs.CV] (ver p. 11).

[32] D. Paper. *Hands-on Scikit-Learn for Machine Learning Applications: Data Science Fundamentals with Python*. Jan. de 2020. ISBN: 978-1-4842-5372-4. DOI: 10.1007/978-1-4842-5373-1 (ver p. 41).

[33] R. Pascanu e T. M. and Yoshua Bengio. "Understanding the exploding gradient problem". Em: *CoRR* abs/1211.5063 (2012). arXiv: 1211.5063. URL: http://arxiv.org/abs/1211.5063 (ver p. 5).

[34] F. Pedregosa et al. "Scikit-learn: Machine learning in Python". Em: *Journal of Machine Learning Research* 12.Oct (2011), pp. 2825–2830 (ver p. 41).

[35] B. Preim e C. P. Botha. *Visual Computing for Medicine, Second Edition: Theory, Algorithms, and Applications*. 2nd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013. ISBN: 0124158730 (ver p. 67).

[36] N. Qian. "On the momentum term in gradient descent learning algorithms". Em: *Neural Networks* 12.1 (1999), pp. 145–151. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(98)00116-6. URL: https://www.sciencedirect.com/science/article/pii/S0893608098001166 (ver p. 38).

[37] S. Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV] (ver p. 19).

[38] D. P. Rivas. *Deep Learning for Beginners*. Packt Publishing Limited, 2020. ISBN: 1838640851 (ver p. 3).

[39] O. Ronneberger, P. Fischer e T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". Em: (2015). arXiv: 1505.04597 [cs.CV] (ver p. 25).

[40] S. Ruder. "An overview of gradient descent optimization algorithms". Em: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747 (ver p. 37).

[41] M. Sewak, M. R. Karim e P. Pujari. *Practical Convolutional Neural Networks*. Packt Publishing Ltd., 2018. ISBN: 978-1-78839-230-3 (ver pp. 4, 6, 7).

[42] J. Shi e J. Malik. "Normalized cuts and image segmentation". Em: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.8 (2000), pp. 888–905. DOI: 10.1109/34.868688 (ver p. 28).

[43] C. Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV] (ver p. 13).

[44] T. Tieleman e G. Hinton. *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning. 2012 (ver p. 39).

[45] T. Tokuoka Y.and Yamada e D. e. a. Mashiko. "3D convolutional neural networks-based segmentation to acquire quantitative criteria of the nucleus during mouse embryogenesis". Em: (2020). DOI: https://doi.org/10.1038/s41540-020-00152-8 (ver p. 11).

[46] S. van der Walt, S. C. Colbert e G. Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation". Em: *Computing in Science Engineering* 13.2 (2011), pp. 22–30. DOI: 10.1109/MCSE.2011.37 (ver p. 41).

[47] I. Vasilev. *Advanced Deep Learning with Python*. Packt Publishing, 2019. ISBN: 9781789956177 (ver p. 7).

[48] C. Vununu et al. "A Deep Feature Learning Scheme for Counting the Cells in Microscopy Data". Em: *2018 IEEE International Conference on Electronics and Communication Engineering (ICECE)*. 2018, pp. 22–26. DOI: 10.1109/ICECOME.2018.8645036 (ver pp. 20, 22).

[49] C.-F. Wang. *A basic introduction to separable convolutions*. Ago. de 2018. URL: https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728 (ver pp. 12, 13, 15).

[50] X. Xia e B. Kulis. "W-Net: A Deep Model for Fully Unsupervised Image Segmentation". Em: *ArXiv* abs/1711.08506 (2017) (ver pp. 7, 16, 26, 28, 29, 31–33, 42, 48, 53).

[51] S. Xie et al. *Aggregated Residual Transformations for Deep Neural Networks*. 2017. arXiv: 1611.05431 [cs.CV] (ver p. 17).

[52] X. Zhang et al. *ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices*. 2017. arXiv: 1707.01083 [cs.CV] (ver pp. 17, 18).

[53] H. Zhou et al. "Image semantic segmentation based on FCN-CRF model". Em: (2016), pp. 9–14. DOI: 10.1109/ICIVC.2016.7571265 (ver p. 19).

[54] Y. Zhu et al. "Improved Prediction on Heart Transplant Rejection Using Convolutional Autoencoder and Multiple Instance Learning on Whole-Slide Imaging". Em: *2019 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)* (2019), pp. 1–4 (ver p. 21).