Experimento U-NET

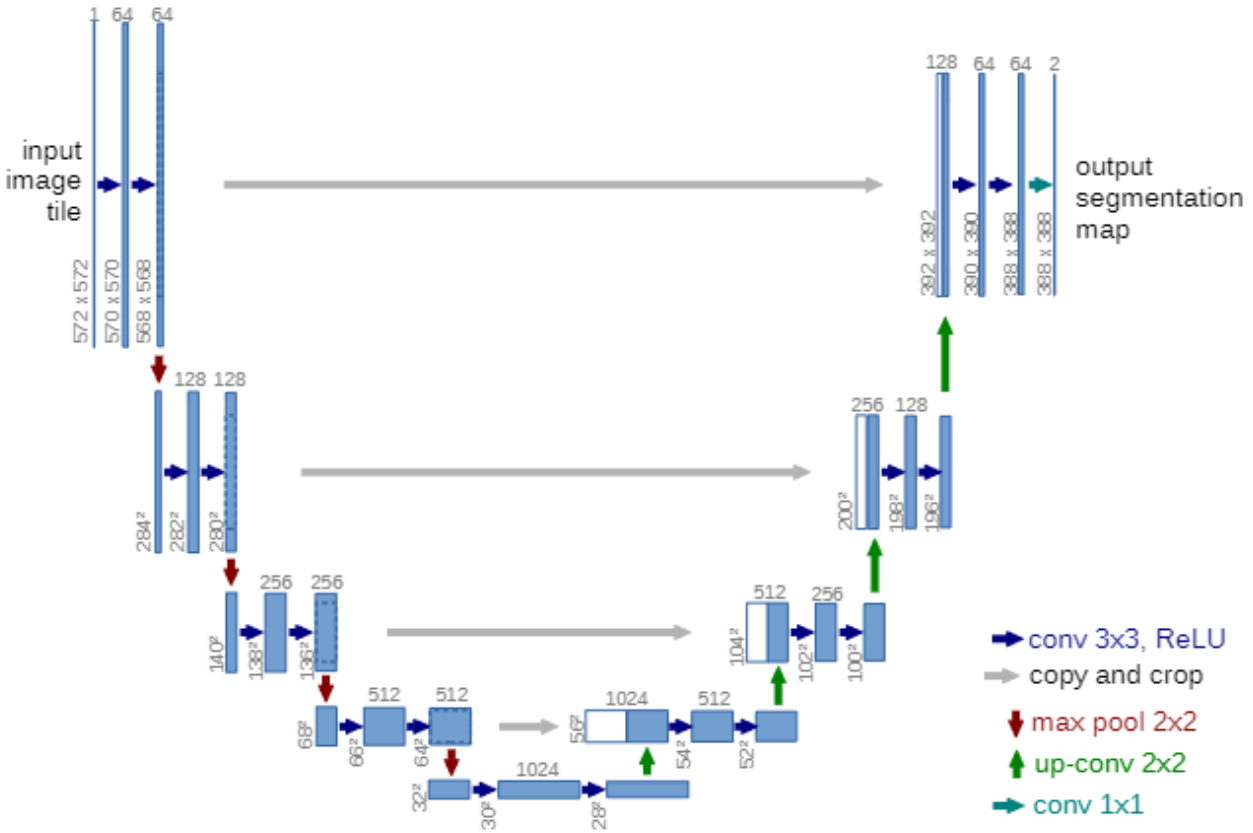
Como acabei não conseguindo explicar as coisas corretamente durante a aula, tentei me redimir escrevendo este texto com o máximo de profundidade e detalhamento que consegui. Peço desculpas pelo uso do português.

Artigo Original

O artigo apresenta uma nova arquitetura de rede neural e uma estratégia de treinamento desenvolvidas especificamente para a segmentação de imagens biomédicas. A motivação para esse trabalho surgiu da seguinte observação: o treinamento bem-sucedido de redes neurais profundas geralmente requer milhares de amostras de treinamento anotadas. Contudo, em tarefas de processamento de imagens biomédicas, a obtenção de um grande número de imagens devidamente anotadas é tipicamente difícil. Além disso, essas tarefas demandam localização precisa, ou seja, a atribuição de um rótulo de classe a cada pixel, em vez de uma única classificação para toda a imagem.

Entre os trabalhos de referência que inspiraram o estudo, destaca-se o de Ciresan et al., que utilizou uma abordagem baseada em janelas deslizantes (sliding windows) para prever o rótulo de cada pixel a partir de patches locais. Embora eficaz, esse método apresentava limitações, sobretudo em termos de velocidade e na conciliação entre a precisão da localização e o uso de contexto.

Com base nessas limitações, os autores desenvolveram uma nova arquitetura, denominada U-Net, que se baseia e expande o conceito de redes totalmente convolucionais (fully convolutional networks). Seu nome deriva da característica forma de U da arquitetura, que reflete a simetria entre as fases de contração e expansão do modelo.



Vou tentar explicar a estrutura geral da rede

Como mencionei antes, a rede tem esse nome por causa do formato em “U” do seu fluxo de dados:

- Lado esquerdo (descendente): caminho de contração (encoder).
- Lado direito (ascendente): caminho de expansão (decoder).
- Centro (parte inferior): o gargalo (bottleneck), que conecta as duas fases.

Encoder

- Cada bloco no lado esquerdo aplica duas convoluções 3×3 seguidas de função de ativação ReLU (parte azul escuro).
- Depois, aplica-se uma operação de max pooling 2×2 (setas vermelhas), que reduz as dimensões espaciais da imagem pela metade, mas duplica o número de canais (ou filtros).
- Isso permite que a rede aprenda características mais abstratas e de maior contexto, mas com menor resolução espacial.

Na imagem temos um exemplo (no topo à esquerda):

- Entrada: 572×572×1
- Após convoluções: 570×570×64, depois 568×568×64
- Após pooling: 284×284×128 (metade da resolução, dobro de filtros)

Bottleneck

No fundo do “U”, a imagem tem dimensões muito reduzidas (28×28), mas muitos canais (1024). Aqui a rede aprende representações altamente abstratas, capturando características globais da imagem.

Decoder

- Cada etapa realiza uma up-convolução (ou transposed convolution) 2×2 (setas verdes), que aumenta a resolução da imagem pela metade e reduz o número de canais.
- Em seguida, há uma concatenação (copy and crop) com a saída correspondente do encoder (setas cinza). Isso recupera informações espaciais finas perdidas na compressão.
- Depois, duas convoluções 3×3 + ReLU novamente refinam o mapa de características.

Camada de Saída

Por fim, há uma convolução 1×1 (seta verde-clara) que reduz os canais para o número de classes desejado (neste caso, 2 → fundo e objeto). O resultado é o mapa de segmentação (output segmentation map), onde cada pixel recebe uma probabilidade de pertencer a cada classe.

Resumo

O encoder extrai recursos (o “o que” da imagem), o decoder reconstrói a forma espacial (o “onde”) e as conexões em espelho preservam detalhes finos e bordas.

Treinamento

A estratégia de treinamento foi concebida para usar poucas amostras anotadas de forma eficiente, provavelmente pela falta de dados para o problema, assim como no nosso desafio do ICASSP. Então no projeto do artigo, foram utilizadas as seguintes considerações:

- Data Augmentation: Usado excessivamente e considerado crucial devido à escassez de dados de treinamento. O método principal foi aplicar deformações elásticas aleatórias às imagens. Isso permite que a rede aprenda a invariança a tais deformações, que são a variação mais comum em tecidos biomédicos.
- Função de Perda Ponderada: Para resolver o desafio de separar objetos da mesma classe que se tocam (como células), os autores introduziram uma função de perda ponderada por pixel. Os rótulos de fundo que separam células em contato recebem um grande peso na função de perda para forçar a rede a aprender as pequenas bordas de separação. O peso ($w(x)$) é calculado com base nas distâncias à borda da célula mais próxima ($d_1(x)$) e da segunda célula mais próxima ($d_2(x)$).
- Otimização: O treinamento utiliza o gradiente descendente estocástico (SGD), implementado no Caffe. É utilizada uma alta taxa de momentum (0.99) e se favorece o uso de grandes blocos de entrada (*input tiles*) em vez de um grande tamanho de lote (*batch size*), que é reduzido a uma única imagem.

Observação: Caffe (Convolutional Architecture for Fast Feature Embedding) é um framework de aprendizado profundo desenvolvido por Yangqing Jia no Berkeley Vision and Learning Center (BVLC), da Universidade da Califórnia, em Berkeley. Foi um dos primeiros frameworks populares de deep learning (lançado em 2014), antes do TensorFlow ou PyTorch, e ficou conhecido pela eficiência e velocidade no treinamento de redes convolucionais (CNNs), especialmente para visão computacional. O Caffe praticamente caiu em desuso por frameworks modernos como TensorFlow (Google), PyTorch (Meta/Facebook) e Keras, mas pelo que vi, na época era o padrão ouro para redes convolucionais.

Resultados e Desempenho

A U-Net demonstrou desempenho superior em várias tarefas de segmentação biomédica:

- **Segmentação de Estruturas Neuronais (EM Stacks):** A U-Net superou o método anterior de janela deslizante. Ela alcançou um erro de *warping* de 0.0003529 (o novo melhor resultado na época, em março de 2015) e um erro *Rand* de 0.0382 no desafio ISBI para segmentação EM.
- **Desafio de Rastreamento de Células ISBI 2015 (Microscopia de Luz):** A rede venceu por uma grande margem em datasets de luz transmitida (contraste de fase e DIC).
    - No dataset **"PhC-U373"**, alcançou IOU (Interseção sobre União) de 92%, superando significativamente o segundo melhor algoritmo (83%).
    - No dataset **"DIC-HeLa"**, alcançou IOU de 77.5%, também superando significativamente o segundo melhor (46%).
- **Velocidade:** A rede é rápida; a segmentação de uma imagem 512x512 leva **menos de um segundo** em uma GPU recente.

Como conclusão, a arquitetura U-Net ofereceu excelente desempenho em diversas aplicações biomédicas, exigindo **muito poucas imagens anotadas** graças à robustez da sua estratégia de aumento de dados com deformações elásticas. O tempo de treinamento é razoável (apenas 10 horas em uma NVidia Titan GPU).

O que diabos o meu código está fazendo então?

Dito tudo isso, vamos entender o que foi que eu fiz.

Estou implementando um experimento para otimizar o treinamento de uma rede U-Net, descrita anteriormente, através do uso de Redução Espacial (Spatial Reduction) a partir de Saliência. Realizo o treinamento da rede em múltiplas iterações e em cada iteração, o modelo é forçado a concentrar-se em uma região progressivamente menor e mais relevante da imagem. Então o fluxo atual é:

1. Treinar o modelo.
2. Usar a previsão do modelo (saliência) para identificar quais pixels são mais importantes para a segmentação (e.g., Top 90%).
3. Zerar os pixels menos importantes, criando um novo conjunto de treino "reduzido".
4. Retreinar o modelo na próxima iteração apenas com os pixels considerados relevantes.

A hipótese proposta foi, ao focar o treinamento nas características essenciais e ignorar o fundo irrelevante de forma progressiva, o modelo pode melhorar a robustez, aumentar a velocidade de convergência, já que estamos treinando em menos dados relevantes e evitar que o modelo aprenda características desnecessárias. É um ciclo de treinamento iterativo e adaptativo.

O primeiro questionamento poderia ser: "Você está usando essa tal de U-net, mas não seria possível usar uma CNN simples para essa tarefa de segmentação?"

Essa é uma ótima pergunta que toca na diferença fundamental entre tarefas de Classificação e Segmentação em Deep Learning, que eu não sabia e por isso fiz até uma tabelinha para lembrar

Tarefa	**Pergunta que responde**	**Saída (Output)**	**Exemplo de uso**
Classificação de Imagens	O que há na imagem?	Um ou mais rótulos	“É um gato siamês.”
Detecção de Objetos	O que há na imagem e onde está localizado?	Caixas delimitadoras + rótulos	Detectar vários animais em uma única foto
Segmentação de Imagens	O que há em cada pixel?	Máscara por pixel	Separar o gato do fundo

Aprendido isso, a resposta pode ser direta: Não! Um modelo CNN padrão não seria adequado para a tarefa de segmentação, pois ele não consegue gerar a saída esperada, que é um mapa de pixels.

A segmentação semântica é uma tarefa de predição densa (pixel-a-pixel).

Característica	**CNN Típica (Ex: VGG, ResNet para Classificação)**	**U-Net / CNN para Segmentação**
Objetivo	Classificar a imagem inteira (Ex: ""É um gato"")	Classificar cada pixel (Ex: ""Este pixel é 'gato', este é 'fundo'"")
Output	Um vetor de classes ([0.9, 0.1]).	Um tensor de imagem com a mesma resolução de entrada (Ex: 128x128x3)
Downsampling	Usa Max Pooling extensivamente, perdendo informações de localização espacial para obter um alto nível de abstração.	Usa Max Pooling, mas compensa essa perda na fase de Decoder (Expansão).

Característica	CNN Típica (Ex: VGG, ResNet para Classificação)	U-Net / CNN para Segmentação
Camada Final	Camada Totalmente Conectada (Fully Connected) que colapsa toda a informação espacial em um vetor.	Camada de Convolução 1x1 que projeta a informação espacial de volta ao número de classes.

Usando uma CNN típica, como eu fiz nos exemplos anteriores, o resultado final seria um único vetor (o rótulo da imagem) e não teria a saída no formato de imagem necessária para zerar os pixels menos importantes.

A U-Net resolve esse problema usando uma arquitetura de Encoder-Decoder com Skip Connections, como vou explicar daqui a pouco.

Explicação do código

Dataset

Oxford-IIIT Pet: https://www.robots.ox.ac.uk/~vgg/data/pets/

Motivo? Nenhum em especial. Achei buscando na internet datasets para tarefas de segmentação. Por algum motivo não havia conseguido acessar o dataset usado no artigo.

Sobre o dataset

O conjunto de dados contém fotografias de cães e gatos com 37 categorias, contendo aproximadamente 200 imagens para cada classe. As imagens apresentam grande variação em escala, pose e iluminação. Todas as imagens possuem anotações de referência (ground truth) associadas, incluindo raça, região de interesse (ROI) da cabeça e segmentação trimap em nível de pixel.

O código foi dividido em partes que irei explicar a seguir:

Preparação dos dados

- `class OxfordPetSegmentation(Dataset)`: É o Dataset que encapsula o conjunto de dados (imagens de pets e suas máscaras de anotação). Responsável por: Extrair os arquivos (.tar.gz), parear imagens com suas máscaras correspondentes, redimensionar as imagens (para 128x128), mapear os valores da máscara (por exemplo, 3 -> 0) e converter a máscara para o formato One-Hot Encoding (cada classe em um canal diferente), que é um formato comum para treinar modelos de segmentação.
- `create_dataloaders`: Divide o dataset total em conjuntos de Treino (80%), Validação (10%) e Teste (10%). Cria os DataLoaders que gerenciam o carregamento dos dados em batches (lotes) e o embaralhamento."
- `get_full_train_data`: Função auxiliar que agrupa todas as imagens e máscaras do dataloader de treino em dois grandes tensores. Isso é feito para que a etapa de redução espacial possa ser aplicada sobre todo o conjunto de treino de uma vez, sem depender dos batches.

Modelo

- `conv_block`: Define o bloco básico de processamento. Aplica duas camadas de Convolução 2D, seguidas pela função de ativação ReLU (Rectified Linear Unit), que introduz não-linearidade no modelo.

- `class UNet(nn.Module)`: Monta a arquitetura completa U-Net. Encoder reduz a dimensão espacial da imagem (usando MaxPool2d) e aumenta a profundidade (número de canais, e.g., de 32 para 64) para capturar características de alto nível (o ""o quê"" é o objeto)." e Decoder aumenta a dimensão espacial de volta ao tamanho original (usando ConvTranspose2d) e diminui a profundidade, reconstruindo a imagem segmentada.
- `Skip Connections`: Conexões essenciais que concatenam características do Encoder com as do Decoder na mesma resolução. Isso permite que o Decoder recupere detalhes espaciais finos que foram perdidos durante a redução do Encoder, melhorando a precisão da segmentação de bordas.

Funções de Saliência e Métricas

- `create_saliency_mask`: Calcula a probabilidade (softmax) e soma as probabilidades das classes de interesse (o que não faz parte do fundo) para obter um mapa de relevância. Define um limiar (e.g., Top 90% dos pixels mais relevantes) e cria uma máscara binária (1 para relevante, 0 para irrelevante)

- `apply_mask`: Aplica a máscara de saliência à imagem original, zerando os pixels não-relevantes (transformando-os em preto/escuro) para a próxima iteração.

- calculate_metrics,iou = intersection / (union + 1e-6),"Calcula as métricas de desempenho para segmentação: mIoU (mean Intersection over Union) e mDice (mean Dice Coefficient). Ambos medem o quanto a previsão do modelo se sobrepõe à verdade fundamental (target), variando de 0 a 1 (quanto mais perto de 1, melhor)."

Sobre o Algoritmo de Saliência

Uma questão que discutimos anteriormente é o motivo do código não utilizar um dos algoritmos de saliência mais conhecidos e complexos, como Grad-CAM ou Integrated Gradients, mas sim um método direto e personalizado, baseado na saída do próprio modelo. O algoritmo de saliência utilizado neste caso é um método de Saliência Baseada na Previsão com Limiar de Percentil.

Cálculo do Score de Relevância (Agregação de Probabilidade)

Esta é a parte que determina o quão "saliência" é um pixel.

- Softmax: O código aplica a função `F.softmax` nas previsões (`predictions`) da rede U-Net. Isso transforma os valores brutos de saída em probabilidades (a soma das probabilidades de todas as classes para um pixel é 1).

- Agregação: Em seguida, ele calcula o `relevance_scores` pela seguinte linha:

```
relevance_scores = torch.sum(probabilities[:, 1:, :, :], dim=1)
```

O `probabilities[:, 1:, :, :]` seleciona todos os canais de classe, exceto o canal de índice 0 (que geralmente representa o "fundo" ou *background* no problema de segmentação). Ao usar o `torch.sum` ao longo do eixo das classes (`dim=1`), o *score* final de relevância de um pixel é a soma das probabilidades de ele pertencer a qualquer classe de objeto de interesse (no caso, o animal ou seu contorno).

Criação da Máscara Binária por Percentil

Em vez de usar um valor fixo, o código utiliza um método adaptativo para definir o limiar.

- **Percentil:** Para cada imagem, o código determina o valor de *score* de relevância que corresponde ao `threshold_percentile` (e.g., 90%).
- **`torch.kthvalue`:** A função `torch.kthvalue` é usada para encontrar o score no percentil desejado (por exemplo, o score do pixel que está no Top 90% dos mais relevantes).
- **Binarização:** Todos os pixels cujo score de relevância **exceda esse valor de limiar** são marcados com **1** (relevante), e os demais são marcados com **0** (irrelevante), criando a máscara binária de saliência.

Parecia ser um método específico e eficaz para o propósito de Redução Espacial Iterativa, ao somar as probabilidades das classes de interesse e ignorar a probabilidade do fundo (classe 0), o algoritmo garante que o mapa de saliência destaque apenas a presença do objeto. O uso do limiar de percentil garante que, independentemente da imagem, o modelo sempre manterá um percentual fixo para a próxima iteração de treinamento, implementando a estratégia de redução espacial de forma controlada.

Treino iterativo

- `train_iteration`: Implementa o ciclo de treino padrão (cálculo de perda, retropropagação, atualização de pesos). Inclui um ReduceLROnPlateau Scheduler que reduz a taxa de aprendizado se a perda de validação parar de melhorar por um certo tempo, ajudando a convergir melhor. Após o treino, executa a avaliação no conjunto de validação para calcular o mIoU e mDice.
- `Loop,"for i in range(1, MAX_ITERATIONS + 1)`: O ciclo que coordena todo o processo: treina o modelo, usa o modelo para identificar os pixels mais relevantes no conjunto de treino, aplica a redução espacial (o apply_mask) e, em seguida, cria um novo dataloader para o próximo ciclo de treinamento, usando as imagens reduzidas.

Resumo

Caso não tenha ficado claro até aqui. O artigo que descreve a arquitetura base que estou usando, no entanto o processo realizado no código não é idêntico ao descrito no artigo, mas sim uma extensão focada na estratégia de treinamento. A principal diferença reside no Ciclo de Treinamento Iterativo com Redução Espacial baseada em Saliência que implementei, o qual não faz parte da metodologia central do artigo original da U-Net. Vou até fazer uma tabela para não perder o costume.

Artigo

Aspecto	**Metodologia U-Net Original**
Arquitetura	Um caminho contrativo (Encoder) para captura de contexto, um caminho expansivo (Decoder) para localização precisa, e Skip Connections para unir ambos.
Treinamento	End-to-End Único: O modelo é treinado em um único ciclo (algumas épocas), usando o conjunto de dados completo (imagens e máscaras) em cada iteração.
Perda (Loss)	Utiliza uma função de perda com mapa de peso de pixels (pixel-wise weight map). Isso é usado para forçar a rede a dar mais importância aos pixels das bordas, ajudando a separar objetos adjacentes, o que é crucial em aplicações biomédicas como a separação de células.
Aumento de Dados	Uso massivo de Data Augmentation (aumento de dados) para simular elasticidade e deformações, crucial quando há poucas amostras de treinamento.

Código do João

Aspecto	Metodologia adotada
Arquitetura	Mesma do artigo
Treinamento	Iterativo com Redução Espacial: O modelo é treinado em vários ciclos (7 iterações), onde o conjunto de dados de treino é modificado a cada ciclo. A rede não é apenas treinada, mas forçada a focar nas regiões mais relevantes identificadas pelo seu próprio mapa de saliência na iteração anterior.
Foco na Saliência (LÁ ELE)	Em vez de um peso estático para as bordas (como no original), seu código usa o resultado da previsão para criar uma máscara de saliência (Baseada na Previsão) e ignorar pixels com baixa probabilidade de serem objetos.

O Marcelo pediu para dar uma olhada nas referências para garantir que não estamos tentando *reinventar a roda*. O que descobri em minhas peregrinações pelo IEEE é que esse processo exato de Redução Espacial Iterativa baseada em Saliência com a metodologia de limiar de percentil de probabilidade (nome grande...) é uma variação específica. O conceito geral de refinar o foco de treinamento de forma iterativa usando mapas de relevância é algo ativamente explorado na pesquisa. Há muitos artigos que buscam o **Refinamento Iterativo e Melhoria da Saliência** em tarefas de segmentação, especialmente em contextos de aprendizado de poucas amostras (Zero-Shot ou Few-Shot) ou detecção de objetos.

1. IteRPrimE: Refinamento Grad-CAM Iterativo Título: [2503.00936] IteRPrimE: Zero-shot Referring Image Segmentation with Iterative Grad-CAM Refinement and Primary Word Emphasis.

Técnica: Este trabalho propõe uma estratégia de refinamento iterativo para segmentação de imagens baseada em texto (Referring Image Segmentation).

Mecanismo: Em vez de usar a probabilidade de softmax diretamente ele usa mapas de Grad-CAM gerados por um modelo Vision-Language. A ideia é a mesma: melhorar progressivamente o foco do modelo na região alvo para superar a imprecisão posicional.

2. Aperfeiçoamento Iterativo de Saliência Título: [2112.00665] Iterative Saliency Enhancement using Superpixel Similarity.

Técnica: O estudo introduz uma abordagem híbrida que iterativamente gera mapas de saliência aprimorados alternando entre a segmentação de superpixels e a estimação de saliência baseada em superpixels.

Mecanismo: O foco aqui é aprimorar o mapa de saliência em si, para que ele se torne mais nítido e preciso, repetindo o processo em ciclos até que os mapas gerados sejam combinados.

3. Redução de Custo Computacional com Saliência Semântica Título: Saliency Prediction with Active Semantic Segmentation.

Técnica: Embora o objetivo principal seja a predição de saliência, o trabalho introduz um modelo de predição de saliência baseado em segmentação semântica ativa.

Mecanismo: O uso de informações de segmentação semântica para orientar e reduzir o subconjunto de regiões a serem processadas para estimar a saliência compartilha a lógica de usar o conhecimento do modelo (segmentação/semântica) para refinar o processamento espacial.

A partir disso, que podemos fazer?