UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

Modelo de Visão Computacional para Identificação de Alimentos em Refeições a partir da Segmentação Obtida por Múltiplas Redes Completamente Convolucionais

Marcos Alberto de Carvalho

Julho de 2024
Itajubá - MG

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

Modelo de Visão Computacional para Identificação de Alimentos em Refeições a partir da Segmentação Obtida por Múltiplas Redes Completamente Convolucionais

Marcos Alberto de Carvalho

Tese submetida ao Programa de Pós-Graduação em Engenharia Elétrica como parte dos requisitos para obtenção do título de **Doutor em Ciências em Engenharia Elétrica.**

Área de concentração: Microeletrônica

Orientador: Prof. Dr. Tales Cleber Pimenta
Coorientadora: Profa. Dra. Alessandra Cristina Pupin Silvério

Julho de 2024
Itajubá - MG

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

Modelo de Visão Computacional para Identificação de Alimentos em Refeições a partir da Segmentação Obtida por Múltiplas Redes Completamente Convolucionais

Marcos Alberto de Carvalho

Banca Examinadora:

Prof. Dr. Tales Cleber Pimenta, UNIFEI (Orientador)
Profa. Dra. Alessandra C. Pupin Silvério, UNIFENAS (Coorientadora)
Prof. Dr. Luiz Eduardo da Silva, UNIFAL
Prof. Dr. Erivelton Antonio dos Santos, UFLA
Prof. Dr. Gabriel Antonio Fanelli de Souza, UNIFEI
Prof. Dr. Bruno Tardiole Kuehne, UNIFEI

Julho de 2024
Itajubá - MG

Aos meus pais, Antônio e Fátima, que sempre acreditaram nos meus sonhos e me ensinaram a ter fé.

Agradecimentos

Agradeço a Deus, que nos protegeu e abençoou durante os desafios que enfrentamos ao longo desta jornada.

Àqueles que foram fundamentais no desenvolvimento deste trabalho, meu sincero agradecimento. Em especial, ao Professor Tales Cleber Pimenta, cuja sabedoria, objetividade e disponibilidade foram essenciais para a realização deste trabalho. Agradeço também à minha coorientadora, Professora Alessandra Cristina Pupin Silvério, que não apenas me apresentou ao problema tratado neste estudo, mas também me motivou de maneira excepcional na busca por soluções, com o nobre propósito de auxiliar as pessoas mais necessitadas.

À UNIFEI, seu corpo docente, o coordenador e os funcionários da pós-graduação, expresso minha sincera gratidão.

Por último, mas não menos importante, quero agradecer à minha esposa, Jaqueline Corrêa Silva de Carvalho, por seu apoio, companheirismo e ajuda incansável ao longo dessa jornada. Aos meus filhos, João Paulo, Germano e Letícia, meu amor incondicional. Amo muito todos vocês.

Resumo

A estratégia de contar os carboidratos dos alimentos consumidos é recomendada por sociedades científicas como uma forma de melhorar a qualidade de vida dos pacientes com diabetes. O monitoramento da ingestão de alimentos pode ser facilitado através do uso de um aplicativo móvel que reconhece automaticamente os alimentos em uma refeição. O reconhecimento automático de imagens de alimentos é considerado uma tarefa desafiadora para a visão computacional devido à semelhança entre os alimentos. Esse desafio aumenta quando o objetivo é classificar alimentos de uma região específica e com um conjunto de dados contendo apenas alimentos dessa região e, portanto, pequeno em comparação com conjuntos de dados públicos de outros países. Para essa tarefa, este trabalho apresenta um modelo que utiliza um conjunto de Redes Completamente Convolucionais (*Fully Convolutional Network* - FCNs) para gerar segmentações dos alimentos em uma refeição. Essas segmentações são processadas por um algoritmo que classifica os alimentos utilizando técnicas de processamento digital de imagens. O modelo tem baixos custos de treinamento e é escalável, ou seja, pode ser treinado para reconhecer um novo alimento sem a necessidade de re-treinar o modelo inteiro. Nos testes, foram utilizados alimentos consumidos no Brasil, obtendo 98.2% de acurácia e 87.8% de sensibilidade.

Abstract

The strategy of counting carbohydrates in consumed foods is recommended by scientific societies as a way to improve the quality of life for diabetes patients. Monitoring food intake can be facilitated by using a mobile application that automatically recognizes the foods in a meal. Automatic recognition of food images is considered a challenging task for computer vision due to the similarity between foods. This challenge increases when the goal is to classify foods from a specific region and with a dataset containing only foods from that region, and therefore, small compared to public datasets from other countries. For this task, this work presents a model that uses a set of Fully Convolutional Networks (FCNs) to generate segmentations of foods in a meal. These segmentations are processed by an algorithm that classifies the foods using digital image processing techniques. The model has low training costs and is scalable, meaning it can be trained to recognize a new food without the need to retrain the entire model. In the tests, foods consumed in Brazil were used, achieving an accuracy of 98.2% and a recall of 87.8%.

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

Segundo a Sociedade Brasileira de Diabetes, atualmente, mais de 13 milhões de pessoas no Brasil estão vivendo com diabetes, o que representa 6,9% da população. As estimativas indicam que até 2040 esse número poderá ultrapassar a marca de 640 milhões em todo o mundo. A Organização Mundial de Saúde estima que a glicemia elevada seja o terceiro fator mais importante na causa de mortalidade prematura, ficando atrás apenas da pressão arterial elevada e do uso do tabaco [1].

Medidas como o estímulo à educação alimentar, atividade física e perda de peso têm um grande impacto positivo, prevenindo complicações e reduzindo as limitações impostas pela doença. A estratégia de contagem de carboidratos dos alimentos ingeridos é recomendada pelas sociedades científicas no Brasil e em todo o mundo como uma forma de melhorar a qualidade de vida dos portadores de diabetes, mantendo a glicemia dentro das concentrações alvo. O tratamento do diabetes mellitus tem como base uma alimentação saudável, que inclui todos os grupos de alimentos. Isso ocorre por meio da relação entre a quantidade adequada de alimentos e sua associação com medicamentos [2].

Com o objetivo de automatizar a contagem de carboidratos, este trabalho apresenta um modelo que utiliza aprendizado profundo e processamento digital de imagens para a classificação automática de alimentos em refeições. Um dos propósitos da inteligência artificial é tornar as atividades mais práticas, automatizadas e interativas, reduzindo os esforços humanos. O processamento digital de imagens envolve a manipulação de imagens em computador com o objetivo de interpretar e classificar os dados presentes na imagem.

1.1 Descrição do problema

Uma maneira eficaz para pessoas com diabetes reduzirem as comorbidades crônicas associadas à doença é monitorar sua ingestão calórica e nutricional. Normalmente, o monitoramento adequado da saúde de indivíduos com diabetes é realizado por meio de um registro do consumo alimentar baseado em observações manuais ou autorrelatos, que podem não revelar com precisão os alimentos consumidos, dificultando assim o cálculo da quantidade real de calorias e carboidratos, bem como a compreensão do comportamento alimentar individual [2][3][4]. As dosagens de medicamentos e insulina são calculadas com base no índice e carga glicêmica dos alimentos ingeridos. Portanto, o conhecimento detalhado da alimentação de pessoas com diabetes em termos de calorias, índice glicêmico e carga glicêmica é crucial para o tratamento e para prevenir complicações como nefropatias, retinopatias, neuropatias e alterações macro e microvasculares, entre outras [1].

O monitoramento da alimentação poderia ser facilitado com o uso de um aplicativo móvel com a capacidade de reconhecer automaticamente os alimentos em uma refeição. No entanto,

o reconhecimento automático de imagens de alimentos é considerado um desafio, uma vez que diferentes alimentos podem ter aparências muito semelhantes, dependendo de como são preparados e apresentados. Essa semelhança resulta em problemas potenciais que afetam diretamente a eficiência dos modelos propostos [5] [6].

Vários trabalhos abordaram esse problema e propuseram modelos para automatizar o reconhecimento de alimentos, utilizando algoritmos de aprendizado profundo treinados com grandes conjuntos de dados públicos, como o FooDD [7], Food101 [8] ou Food-5K [9], que contêm exemplos de alimentos e modos de preparo característicos de regiões específicas [10] [11]. Por exemplo, o banco de dados UEC-Food256 [12] combina um grande número de imagens de alimentos japoneses, o que significa que um modelo treinado com esse banco não identificaria alimentos de outras regiões ou países. Para obter um modelo de reconhecimento de alimentos mais preciso e universal, seria necessário construir conjuntos de dados maiores contendo imagens de alimentos de todo o mundo. Isso, no entanto, requer um alto investimento de tempo tanto na preparação dos dados quanto no treinamento do modelo.

Portanto, uma abordagem viável para o desenvolvimento de um sistema de reconhecimento automático de alimentos que gere informações significativas para o público-alvo é a utilização de conjuntos de treinamento com alimentos que façam parte da dieta comum desse público. A principal dificuldade nesse caso está relacionada ao volume de dados disponíveis para o treinamento do modelo, que pode ser relativamente pequeno em comparação com os conjuntos de dados normalmente utilizados em aprendizado profundo.

1.2 Objetivos

Vários estudos têm proposto modelos para classificar automaticamente imagens de alimentos. No entanto, os modelos voltados para a identificação de alimentos brasileiros são limitados. Portanto, o objetivo desta pesquisa foi desenvolver um modelo para identificar alimentos brasileiros, permitindo que aplicativos móveis determinem a quantidade de carboidratos, o que é essencial para o controle do diabetes. Essa é uma tarefa desafiadora para a visão computacional devido a variação na aparência e a semelhança entre esses alimentos, resultando em desafios que afetam diretamente a eficiência dos modelos propostos [13][14][15]. A complexidade aumenta quando o objetivo é classificar diversos alimentos em uma mesma refeição.

Para atingir os objetivos, foram definidos os seguintes objetivos específicos:

- Criar de um conjunto de treinamento contendo alimentos brasileiros.
- Definir o pré-processamento necessário das imagens que compõem o conjunto de treinamento. Isso inclui a rotulagem dos alimentos a serem classificados.
- Definir a arquitetura de rede neural que serão utilizadas na tarefa de segmentação dos alimentos.

- Desenvolver um algoritmo de classificação de alimentos que, a partir das segmentações geradas para uma refeição, processe essas segmentações e identifique os alimentos presentes na refeição considerando a possibilidade de múltiplas segmentações para o mesmo alimento.
- Implementar e avaliar da eficiência do modelo proposto.

1.3 Contribuições

A seguir estão listadas algumas das contribuições da pesquisa desenvolvida:

- Desenvolvimento de um modelo capaz de identificar com precisão alimentos brasileiros, o que pode facilitar o monitoramento dietético de diabéticos que precisam controlar a ingestão de carboidratos.
- Abordagem baseada em redes completamente convolucionais e em um Algoritmo de Identificação de Alimentos, que também pode ser útil para identificar alimentos em outras regiões do mundo com características semelhantes aos alimentos brasileiros.
- Construção e disponibilização de um banco de dados com imagens de refeições contendo alimentos brasileiros e suas respectivas máscaras de segmentação.

1.4 Hipóteses

As seguintes hipóteses foram examinadas neste estudo. A primeira hipótese considera que a utilização de um conjunto de redes completamente convolucionais, cada uma treinada para segmentar um alimento específico, proporciona a precisão necessária para a identificação dos alimentos, resolvendo o problema da semelhança entre eles.

A segunda hipótese é que o modelo proposto é escalável, ou seja, a inclusão de novos alimentos na base de treinamento mantém a rapidez na execução do aprendizado e também a precisão na identificação dos alimentos.

1.5 Estrutura do trabalho

O trabalho está organizado em cinco capítulos. No primeiro capítulo, é apresentada uma visão geral do assunto abordado durante o desenvolvimento desta tese, bem como os objetivos e hipóteses da pesquisa. No segundo capítulo, são apresentados trabalhos relacionados a este estudo, seguidos de uma fundamentação teórica nas principais áreas temáticas envolvidas no trabalho. No terceiro capítulo, o modelo proposto é apresentado em detalhes. No quarto capítulo, os resultados experimentais são apresentados e discutidos. Por fim, no quinto capítulo, as conclusões desta pesquisa são apresentadas.

2 Fundamentação teórica

Neste capítulo, serão apresentados trabalhos relacionados a este estudo, seguidos de uma fundamentação teórica nas principais áreas temáticas envolvidas no trabalho, como processamento digital de imagens, fundamentos da inteligência artificial e conceitos básicos de redes neurais artificiais. Também será abordada a visão computacional e sua implementação por meio das redes neurais convolucionais e completamente convolucionais. Por fim, algumas métricas de avaliação serão apresentadas.

2.1 Trabalhos relacionados

Métodos de classificação de imagens de alimentos baseados em Redes Neurais Convolucionais (*Convolutional Neural Network* - CNN) são amplamente utilizados em estudos com o objetivo de desenvolver classificadores de alimentos. No entanto, ao buscar trabalhos relacionados, ficou evidente que a maioria utilizam os mesmos bancos de dados públicos. A seguir, alguns desses trabalhos são apresentados.

Kagaya *et al.* [16] desenvolveram um sistema de reconhecimento baseado em CNN para detecção e segmentação de diferentes alimentos em imagens. Os autores constataram que os núcleos de convolução desempenham um papel significativo na extração de características, com a cor sendo um fator dominante. Ao treinarem o modelo com o conjunto de dados UEC-256, conseguiram atingir uma precisão de 87%.

Myers *et al.* [10] aplicaram uma CNN para segmentar e classificar alimentos em imagens, criando um aplicativo móvel que permite aos usuários fotografar suas refeições. O sistema identifica automaticamente os alimentos presentes na imagem, estima o tamanho da porção e calcula calorias e nutrientes. Utilizando o conjunto de dados Food101 no treinamento, alcançaram uma precisão de 76%.

Liu *et al.* [17] apresentaram um sistema de reconhecimento de alimentos baseado em avaliação dietética, implementado em uma infraestrutura de serviço de computação de borda para redução de latência e uso de recursos computacionais. O sistema alcançou uma precisão de 87% na identificação correta dos alimentos.

Reddy *et al.* [5] combinaram os conjuntos de dados públicos FooDD e Food101 para criar um conjunto de dados para classificar 20 classes de alimentos. Usando redes completamente convolucionais, alcançaram uma precisão de 78,7%.

Islam *et al.* [18] utilizaram o conjunto de dados Food-11 com 11 classes de alimentos. As imagens foram pré-processadas para treinar um modelo de rede neural convolucional com camadas de convolução e *pooling*, bem como camadas completamente conectadas para a classificação final. O modelo proposto alcançou uma precisão de 74,7%, enquanto o uso de um modelo pré-treinado resultou em uma precisão de 83,5%.

Samraj *et al.* [19] desenvolveram um modelo de classificação para 17 classes de alimentos usando um conjunto de dados com 170 imagens. Já Memis *et al.* [20] conduziram um estudo comparativo sobre o desempenho de vários métodos de aprendizado profundo no reconhecimento de imagens de alimentos. Os experimentos foram realizados com o conjunto de dados UEC Food-100, utilizando a abordagem de transferência de aprendizado, com todos os modelos treinados com pesos pré-definidos do ImageNet. O melhor resultado de classificação alcançou uma precisão de 87,7%.

Alguns países têm poucos estudos dedicados à identificação de alimentos, como é o caso do Brasil. Isso pode ser atribuído à falta de conjuntos de dados públicos significativos para esses países. Poucos exemplos dessa abordagem incluem o trabalho de Islam *et al.* [18], que se concentra em alimentos australianos, Ege e Yanai [21], que abordam alimentos japoneses e americanos, e Chun *et al.* [15], que se dedicam a alimentos coreanos. Em relação a alimentos brasileiros, Freitas *et al.* [14] e Shiga [22] representam os únicos exemplos de pesquisa disponíveis. É importante notar que todos esses estudos envolveram a criação de conjuntos de dados específicos para treinar os modelos correspondentes.

Com base no exposto, tornou-se evidente porque um dos objetivos deste trabalho foi a criação de um conjunto de dados contendo alimentos brasileiros. Somente assim foi possível desenvolver um modelo apropriado capaz de classificar esses alimentos corretamente.

Os próximos tópicos apresentam os fundamentos teóricos subjacentes ao desenvolvimento do modelo proposto.

2.2 Processamento digital de imagens

O processamento digital de imagens permite o desenvolvimento de aplicações que melhoram as informações contidas nas imagens para interpretação humana e também para a interpretação automática em sistemas de visão computacional. De acordo com Gonzalez e Woods [23], no processamento digital de imagens, as entradas e saídas são imagens e envolvem processos de extração de atributos até o reconhecimento de objetos individuais.

2.2.1 Modelo de imagens

A representação e manipulação de uma imagem em um computador requer a definição de um modelo matemático adequado da imagem. Uma imagem pode ser definida como uma função de intensidade luminosa, denotada por $f(x, y)$, em que o valor ou amplitude nas coordenadas espaciais (x, y) fornece a intensidade ou brilho da imagem naquele ponto. A Figura 2.1 mostra uma imagem e a orientação do sistema de coordenadas. A origem de uma imagem digital está localizada na parte superior esquerda ($f(0, 0)$), com o eixo x positivo se estendendo para baixo e o eixo y positivo se estendendo para a direita. Os dispositivos de visualização de imagem, como monitores de TV e de computadores, varrem uma imagem começando no canto superior esquerdo e movendo-se para a direita, uma linha por vez.

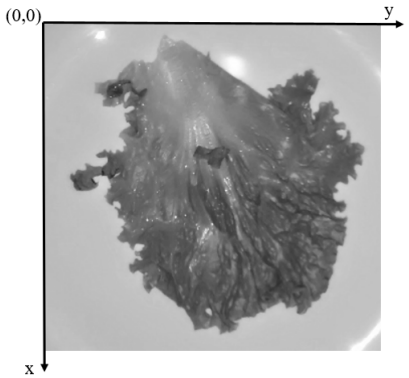


Figura 2.1 – Conversão do sistema de coordenadas para representação de imagens digitais [22].

2.2.2 Digitalização

A função $f(x, y)$ deve ser convertida para uma forma discreta. Uma imagem digital pode ser obtida por um processo denominado *digitalização*, que envolve dois passos: amostragem e quantização.

A amostragem consiste em discretizar o domínio de definição da imagem nas direções x e y, gerando, assim, uma matriz $M \times N$ de amostras, respectivamente. A quantização, por sua vez, envolve a escolha de um número inteiro L de níveis de cinza (para imagens em escala de cinza) para cada ponto da imagem.

Cada elemento $f(x, y)$ dessa matriz é chamado de pixel, com $0 \leq x \leq M-1$ e $0 \leq y \leq N-1$. A imagem contínua $f(x, y)$ é, portanto, aproximada por uma matriz de dimensão M pixels na vertical (eixo x na Figura 2.1) e N pixels na horizontal (eixo y na Figura 2.1).

A representação de uma matriz $M \times N$ pode ser escrita na forma da Equação 2.1.

$$f(x, y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix} \tag{2.1}$$

2.2.3 Representação de imagens digitais

Uma imagem digital pode ser representada por meio de uma matriz bidimensional, em que cada elemento da matriz corresponde a um pixel da imagem. A Figura 2.2 mostra a representação matricial de uma imagem, com uma pequena região destacada, composta por números inteiros que correspondem aos níveis de cinza dos pixels da imagem.

Para fins computacionais e no desenvolvimento de algoritmos, é útil ajustar os valores de intensidade L para o intervalo $[0, 1]$. Nesse caso, eles deixam de ser números inteiros. No entanto, na maioria dos casos, esses valores são novamente ajustados para o intervalo de números inteiros $[0, 255]$ para fins de armazenamento e exibição.

193	193	192	188	184	178	171	165	158	149	141	136	133	134	133
192	188	182	175	169	163	155	150	147	141	134	129	124	119	113
181	173	162	154	148	143	136	132	126	124	120	117	111	102	95
161	152	141	136	132	128	123	119	106	102	96	92	89	85	90
145	137	131	128	125	118	111	107	92	86	78	76	79	83	100
132	127	123	121	116	105	95	89	82	80	80	89	102	113	135
121	119	114	109	102	91	82	79	81	86	94	107	128	151	169
113	114	107	96	85	79	80	86	94	105	121	138	158	176	188
106	105	95	80	73	76	90	103	119	133	152	169	181	190	194
101	95	84	79	85	99	118	133	152	164	178	188	192	193	194
94	89	87	96	116	134	151	165	175	182	190	194	194	195	197
94	98	108	126	147	162	175	185	187	190	193	194	193	193	195
114	128	143	157	171	181	188	193	196	196	196	196	195	194	193
141	159	173	179	186	193	196	197	196	195	194	195	196	195	193
175	182	190	194	196	196	196	195	192	194	197	196	197	193	199

(a) (b)

Figura 2.2 – Representação matricial. (a) imagem; (b) níveis de cinza correspondente a região em destaque.

2.2.4 Alguns relacionamentos básicos entre pixels

Neste tópico, serão apresentadas relações importantes entre pixels em uma imagem digital. Como mencionado anteriormente, uma imagem é representada por $f(x, y)$. Neste contexto, um pixel específico será denotado pelas letras p e q.

2.2.4.1 Vizinhos de um pixel

Um pixel p nas coordenadas (x, y) possui quatro vizinhos horizontais e verticais cujas coordenadas são $(x + 1, y)$, $(x - 1, y)$, $(x, y + 1)$ e $(x, y - 1)$. Esses pixels formam a chamada *vizinhança*-4 de p, que é representada por $N_4(p)$. Os quatro vizinhos diagonais de p têm coordenadas $(x - 1, y - 1)$, $(x - 1, y + 1)$, $(x + 1, y - 1)$ e $(x + 1, y + 1)$, constituindo o conjunto $N_d(p)$. Esses pontos, juntamente com a *vizinhança*-4, são conhecidos como *vizinhança*-8 de p, representada por $N_8(p)$ [23]. A Figura 2.3 ilustra o pixel p com as *vizinhança*-4 e *vizinhança*-8.

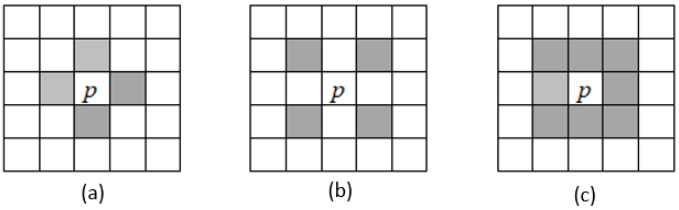


Figura 2.3 – Tipos de vizinhança. (a) *vizinhança*-4 de um pixel. (b) Vizinhos diagonais de um pixel. (c) *vizinhança*-8 de um pixel.

2.2.4.2 Conectividade

A vizinhança de um pixel é um conceito importante para verificar a *conectividade* entre pixels, a fim de identificar limites de objetos e componentes de regiões em uma imagem. Para

determinar se dois pixels são conectados, é necessário verificar se eles são vizinhos, conforme o tipo de vizinhança adotado, e se os pixels atendem a critérios específicos de similaridade, como intensidade de cinza, cor ou textura. Em uma imagem binária, por exemplo, em que os pixels podem assumir os valores 0 ou 1, dois pixels podem ter uma vizinhança-4, mas só serão considerados conectados se tiverem o mesmo valor [24].

2.2.4.3 Rotulação

A *rotulação* de componentes conexos é um procedimento que tem como objetivo atribuir um *rótulo* a cada objeto (ou componente conexo) na imagem. O algoritmo de rotulação transforma uma imagem binária em uma imagem simbólica que representa os objetos conectados, sendo que cada um deles possui um rótulo específico [25].

Na Figura 2.4(a), é apresentada uma imagem bidimensional binária. Na Figura 2.4(b), a rotulação foi realizada considerando a vizinhança-4, resultando em uma imagem com dois componentes conexos, cada um com um rótulo diferente. Por outro lado, na Figura 2.4(c), a rotulação foi feita considerando a vizinhança-8, resultando em apenas um componente conexo com um único rótulo atribuído.

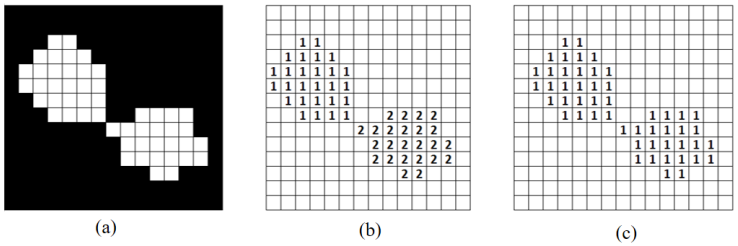


Figura 2.4 – Componentes conexos. (a) Imagem binária. (b) Vizinhança-4, dois componentes. (c) Vizinhança-8, um componente.

2.2.4.4 Algoritmo de rotulação

Para implementar a rotulação de componentes conexos, existem vários algoritmos apresentados na literatura. Neste contexto, será apresentado um dos algoritmos de rotulação mais simples disponíveis.

No algoritmo apresentado, uma imagem pode ser representada como um grafo, onde os pixels são os vértices do grafo. Um grafo é uma estrutura de dados composta por um conjunto de pontos, chamados de vértices, que são conectados por um conjunto de linhas, chamadas de arestas. Portanto, uma imagem pode ser representada como um grafo, onde os pixels correspondem aos vértices e as relações de adjacência entre os pixels são representadas pelas arestas [26].

Dessa forma, é possível explorar algoritmos eficientes em grafos para rotular os pixels de uma imagem. O algoritmo apresentado é o da busca em largura, cujo pseudocódigo está ilustrado na Figura 2.5.

```
Inicialize a fila F
novo_rótulo ← 1
Para todo pixel p de uma imagem, tal que rótulo(p) = 0
        rótulo(p) ← novo_rótulo e insira p na fila F
        Enquanto a fila F não estiver vazia
                Remova p da fila F
                Para todo q vizinho de p, tal que rótulo(q) = 0
                        rótulo(q) ← novo_rótulo e insira q na fila F
        Incremente novo_rótulo
```

Figura 2.5 – Algoritmo para rotulação de componentes conexos.

Para implementar a busca em largura, é necessária uma estrutura de dados do tipo fila que segue o princípio FIFO (*First In*, *First Out*). Nessa estrutura, os elementos recém-adicionados são inseridos no final da fila e as remoções ocorrem no início da fila.

2.2.4.5 Operações lógicas com imagens binárias

Quando se trabalha com imagens binárias, adota-se a terminologia em que os pixels de *fundo* têm valor 0 (preto) e os pixels de *frente* têm valor 1 (branco), representando os objetos nas imagens. Ao lidar com imagens binárias, é comum referir-se à interseção, união e complemento como as operações lógicas E (AND), OU (OR) e NÃO (NOT), onde o termo "lógicas" deriva da lógica matemática, em que 1 e 0 representam verdadeiro e falso [23][24].

A operação AND produz um valor 1 na imagem resultante quando os pixels correspondentes nas duas imagens de entrada têm um valor igual a 1. O resultado da operação OR é 1 quando pelo menos um dos pixels nas imagens é igual a 1. A operação NOT inverte o valor do pixel na imagem.

As operações lógicas podem ser usadas para combinar informações entre imagens ou extrair regiões de interesse. Alguns exemplos de aplicação das operações lógicas são mostrados na Figura 2.6.

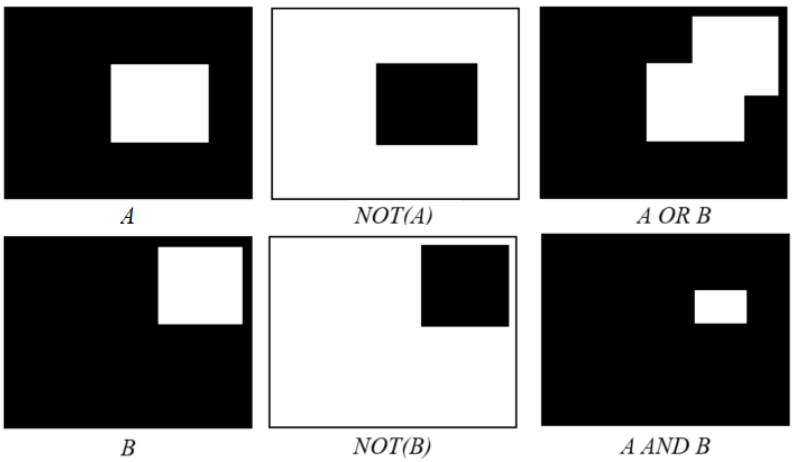


Figura 2.6 – Operações lógicas em imagens binárias.

2.2.5 Filtragem espacial

Neste tópico, serão apresentados alguns dos conceitos básicos que fundamentam a utilização de filtros espaciais no processamento de imagens. Essas técnicas têm como objetivo remover ruídos, suavizar ou realçar determinadas regiões da imagem. Como exemplo, considere a Figura 2.7(a), que exibe uma imagem com alguns ruídos. O resultado da aplicação de um filtro é mostrado na Figura 2.7(b). Ao comparar o resultado com a imagem original, pode-se verificar que ele representa de forma razoável os objetos na imagem.

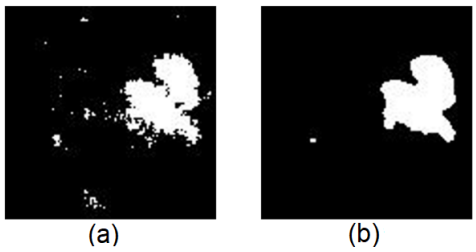


Figura 2.7 – Remoção de ruído. (a) Imagem com ruídos. (b) Imagem após aplicação de um filtro da mediana 5×5.

Os processos no domínio espacial, que serão apresentados neste tópico, podem ser expressos da seguinte forma [23]:

$$g(x, y) = T[f(x, y)] \tag{2.2}$$

Aqui, $f(x, y)$ representa a imagem de entrada, $g(x, y)$ é a imagem de saída, e T é um operador definido em uma vizinhança do ponto (x, y). A Figura 2.8 ilustra a aplicação básica da Equação 2.2 em uma imagem. O ponto (x, y) mostrado é uma posição arbitrária na imagem, e a pequena região que contém o ponto é a vizinhança de (x, y).

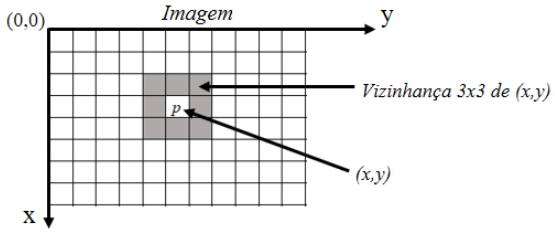


Figura 2.8 – Uma vizinhança 3×3 ao redor de um ponto (x, y) em um domínio espacial.

O processo, conforme ilustrado na Figura 2.8, envolve mover a origem da vizinhança de um pixel para outro e aplicar o operador T aos pixels na vizinhança para gerar a saída naquela posição. Portanto, para qualquer posição específica (x, y), o valor da imagem de saída g nessas coordenadas é igual ao resultado da aplicação de T à vizinhança com origem (x, y) na imagem f. Esse procedimento é conhecido como filtragem espacial, onde a vizinhança, junto com a operação predefinida, é chamada de filtro espacial (também conhecido como *kernel*). O tipo de operação realizada na vizinhança determina a natureza do processo de filtragem [23].

A filtragem cria um novo pixel com coordenadas iguais às coordenadas do centro da vizinhança, e o valor desse novo pixel é o resultado da operação de filtragem. Uma imagem processada (filtrada) é gerada à medida que o centro do filtro percorre cada pixel na imagem de entrada. Se a operação realizada sobre os pixels da imagem for linear, o filtro é chamado de *filtro linear*. Caso contrário, o filtro é não linear [23][24].

A Figura 2.9 ilustra o funcionamento da filtragem espacial linear usando uma vizinhança 3×3. Em qualquer ponto (x, y) da imagem, a resposta, $g(x, y)$, do filtro é a soma dos produtos dos coeficientes do filtro com os pixels da imagem abrangidos pelo filtro [23]:

$$\begin{aligned} g(x,y) = w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y) + \\ \ldots + w(0,0)f(x,y) + \ldots + w(1,1)f(x+1,y+1) \end{aligned} \tag{2.3}$$

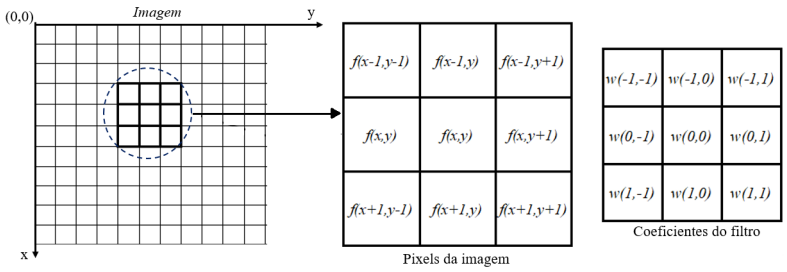


Figura 2.9 – Filtragem linear utilizando uma máscara 3×3.

O coeficiente central do filtro, $w(0, 0)$, alinha-se com o pixel na posição (x, y). Para um tamanho de máscara $m \times n$, onde $m = 2a + 1$ e $n = 2b + 1$, com a e b sendo números inteiros positivos os filtros terão tamanho ímpar, com o menor tamanho sendo 3×3. Em geral, a filtragem espacial linear de uma imagem de dimensões $M \times N$ com um filtro de dimensões $m \times n$ é dada pela expressão [23]:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} [w(s,t)f(x+s,y+t)] \tag{2.4}$$

2.2.5.1 Convolução espacial

A convolução pode ser entendida como a aplicação de um filtro e consiste em mover uma máscara pela imagem e calcular a soma dos produtos em cada posição, como explicado no tópico anterior e dada pela Equação 2.4.

O aspecto importante é que a máscara do filtro usada em uma tarefa de filtragem deve ser especificada para corresponder à operação desejada [23]. Dependendo das dimensões e dos valores presentes no filtro, é possível obter diferentes características nas imagens, como suavização, realce de bordas, deslocamento, contraste, texturas, remoção de ruídos, entre outras.

A Figura 2.10 apresenta dois filtros usados na detecção de bordas, conforme descrito em [23]. A Figura 2.11 mostra a aplicação desses dois filtros em uma imagem. Na Figura 2.11(b), pode-se

observar o resultado obtido pelo filtro Laplaciano, que resulta na detecção de bordas. Por outro lado, na Figura 2.11(c), o filtro de Sobel é aplicado, produzindo fronteiras bem definidas entre os objetos, destacando os contornos.

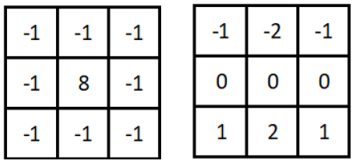


Figura 2.10 – (a) Laplaciano (b) Operadores de Sobel.

Figura 2.11 – (a) Imagem original. (b) Aplicação do filtro Laplaciano. (c) Resultado da aplicação do filtro Operadores de Sobel.

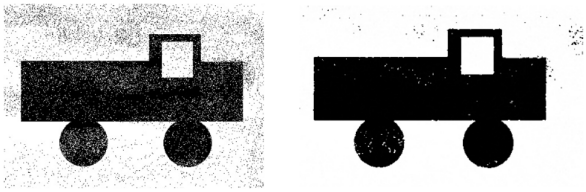
Na literatura, são encontrados termos como *filtro de convolução*, *máscara de convolução* ou *kernel de convolução* para se referir a um filtro espacial utilizado em processamento de imagens.

2.2.6 Filtros não lineares

Uma classe de filtros não lineares amplamente empregados em processamento de imagens é composta pelos *filtros estatísticos de ordem*. São filtros espaciais cuja resposta baseia-se na ordenação (classificação) dos pixels contidos na área da imagem coberta pelo filtro, substituindo o valor do pixel central pelo valor determinado pelo resultado da classificação. Um dos filtros dessa categoria é o *filtro da mediana*, que substitui o valor de um pixel pela mediana dos valores de intensidade na vizinhança desse pixel. Os filtros de mediana são eficazes na presença de ruído impulsivo, também chamado de *ruído sal e pimenta*, devido à sua aparência, como pontos brancos e pretos sobrepostos em uma imagem [23].

A Figura 2.12 apresenta uma imagem com ruído sal e pimenta e o resultado da aplicação do filtro da mediana, respectivamente.

A filtragem pela mediana em um ponto da imagem ocorre por meio da ordenação dos pixels sob a máscara. Por exemplo, em uma vizinhança 3×3, a mediana é o quinto valor; em uma vizinhança 5×5, é o 13° valor. Dessa forma, a principal função dos filtros de mediana é tornar os pontos com níveis de intensidade distintos mais semelhantes aos seus vizinhos.



(a) (b)

Figura 2.12 – Aplicação de filtro não linear. (a) Imagem corrompida pelo ruído sal e pimenta. (b) Redução de ruído com um filtro de mediana 3×3.

2.2.7 Processamento de imagens coloridas

A utilização da cor no processamento de imagens coloridas é motivada principalmente pelo fato de que a cor é um poderoso descritor que pode simplificar a identificação do objeto em sua extração de uma imagem. Por exemplo, em [5], o reconhecimento automático de alimentos é implementado a partir de imagens coloridas para extração de características, visando maior precisão. Em [4], [6], [10] e [27] também são utilizadas imagens coloridas com o mesmo propósito.

Em termos de processamento digital de imagens, o modelo mais utilizado na prática é o modelo RGB. Esse modelo combina as cores primárias vermelha (R, *red*), verde (G, *green*) e azul (B, *blue*) [24]. Baseia-se em um sistema de coordenadas cartesianas. O subespaço de cores de interesse é o cubo, apresentado na Figura 2.13, no qual os valores RGB primários estão entre os vértices; as cores secundárias ciano, magenta e amarelo estão em outros três vértices; o preto está na origem; e o branco está no vértice mais distante da origem. Nesse modelo, a escala de cinza estende-se do preto até o branco ao longo do segmento de reta que une esses dois pontos.

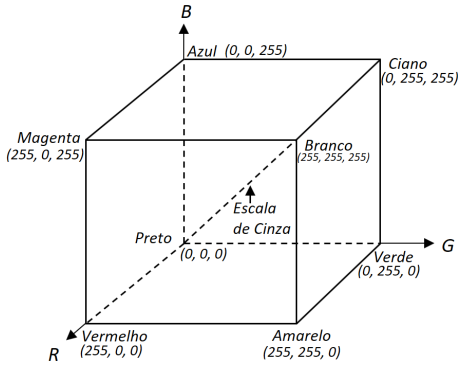


Figura 2.13 – Esquema do cubo de cores RGB [23].

Imagens representadas no modelo de cores RGB consistem de três componentes de imagens, uma para cada cor primária. A Figura 2.14(a) mostra uma representação espacial de uma imagem 10×10 pixels. Nessa imagem, existem pixels vermelhos, azuis, verdes e brancos. As Figuras 2.14(b), (c) e (d) mostram as três componentes de imagem individuais correspondentes a cada pixel colorido. Nas componentes de imagem, o valor 0 (zero) representa preto e 255 representa branco. Os componentes R, G e B são, então, individualmente, imagens em escala de cinza. Essas três imagens se combinam para produzir uma imagem em cores.

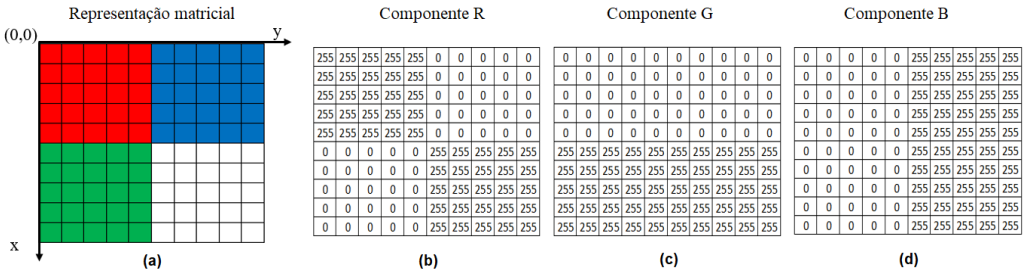


Figura 2.14 – Imagem 10×10 pixels colorida e componentes R, G e B.

Como mencionado anteriormente, a cor é um poderoso descritor que pode simplificar a identificação do objeto em uma imagem. Em sistemas de visão computacional, foco deste trabalho, as imagens utilizadas como entrada são coloridas, uma vez que a cor é um atributo importante que diferencia os alimentos. Somente a forma, contornos ou intensidades de cinza não são atributos suficientes para caracterizar um alimento específico.

2.3 Rede Neurais Artificiais

Os computadores são máquinas capazes de executar programas. Um programa é a codificação de um algoritmo utilizando uma linguagem de programação. Um algoritmo, por sua vez, é a sequência finita de passos da solução formal do problema. Entretanto, alguns tipos de problemas não possuem solução algorítmica, isto é, não é possível escrever programas para resolvê-los. Alguns exemplos desses problemas incluem o reconhecimento de imagens em geral, faces humanas, direção de automóveis ou jogos de *Go*, entre muitos outros. O reconhecimento de alimentos é uma dessas tarefas consideradas desafiadoras devido à semelhança entre as classes de alimentos.

Para abordar problemas sem solução algorítmica, podem ser utilizadas técnicas conhecidas como aprendizado de máquina. Essas técnicas permitem que os computadores aprendam a partir de um conjunto de exemplos, ou seja, através de dados. Esse método de aprendizado é semelhante ao processo de aprendizagem humano.

O aprendizado de máquina pode ser categorizado em aprendizado supervisionado, não supervisionado e por reforço. O aprendizado supervisionado envolve um conjunto de dados no qual a classe de cada exemplo é conhecida, ou seja, são dados rotulados. No aprendizado não supervisionado, os dados não possuem rótulos, e o aprendizado ocorre por meio de processos iterativos que analisam os dados sem intervenção humana. Nesse caso, o aprendizado é realizado por meio de métodos de clusterização e associação. O aprendizado por reforço é baseado em um algoritmo que não recebe a resposta correta, mas sim um sinal de reforço, recompensa ou punição. O algoritmo faz uma hipótese com base nos exemplos e determina se essa hipótese foi boa ou ruim. O aprendizado por reforço é amplamente utilizado em jogos e robótica [28] [29] [30] [31] [32].

Uma das técnicas de aprendizado supervisionado são as Redes Neurais Artificiais (RNAs).

As RNAs são modelos matemáticos inspirados nas estruturas neurais biológicas e adquirem capacidade computacional por meio do aprendizado. O processamento da informação em uma RNA é realizado por neurônios artificiais, que estão conectados por conexões sinápticas, cada uma associada a um peso.

O interesse em redes neurais remonta ao início dos anos 1940, com o trabalho de McCulloch e Pitts em 1943 [33]. Eles propuseram modelos de neurônios na forma de dispositivos de limiarização binária e algoritmos estocásticos envolvendo mudanças abruptas entre os estados dos neurônios, como base para o modelamento dos sistemas neurais.

A Figura 2.15 mostra o modelo matemático de um neurônio denominado de *perceptron*, pesquisado e desenvolvido por Rosenblatt [34] no início dos anos 1960, que resolve problemas simples que são linearmente separáveis.

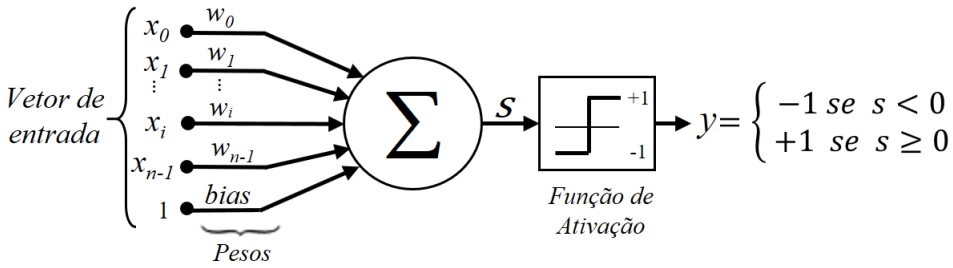


Figura 2.15 – Modelo de *perceptron*.

As entradas do neurônio recebem um vetor com os valores correspondentes aos padrões a serem classificados, como valores de pixels de imagem ou características de um objeto. A resposta desse dispositivo básico é baseada em uma soma ponderada de suas entradas, ou seja,

$$s = \sum_{i=0}^{n} (w_i x_i) + bias \tag{2.5}$$

Os coeficientes $w_i, i = 0, 1, \ldots, n-1$, e *bias* são chamados pesos e representam o aprendizado da rede. Eles modificam as entradas antes de serem somadas e introduzidas na função de ativação (Seção 2.3.1). O *bias* tem entrada com valor fixo e é um peso utilizado para ajustes finos, aumentando o grau de liberdade dos ajustes dos pesos durante o treinamento [35].

A saída do neurônio é o resultado da aplicação de uma função de ativação que recebe o somatório conforme a Equação 2.5. Quando $S < 0$, a função de ativação faz com que a saída y do *perceptron* seja -1, indicando a classe do vetor de características presente na entrada. Da mesma forma, quando $S \geqslant 0$, a função de ativação faz com que a saída y seja $+1$, indicando que a entrada pertence a outra classe. Essa é uma função de decisão simples para duas classes de padrões.

O algoritmo de aprendizagem do *perceptron* utiliza a correção de erros (diferença entre a resposta desejada e a resposta da rede) como base. O processo de aprendizagem envolve uma fase de treinamento e uma fase de teste do algoritmo. Na fase de treinamento, exemplos

rotulados são apresentados ao algoritmo, e os parâmetros da rede (pesos) são modificados a cada apresentação de um novo exemplo. Na fase de teste, o sistema é avaliado.

No entanto, o modelo do *perceptron* está limitado a resolver problemas linearmente separáveis, nos quais a fronteira de decisão toma a forma de uma linha reta. Isso é ilustrado na Figura 2.16, na qual consideramos a função lógica "AND", em que dois bits de entrada são representados como pontos no espaço. Nesse caso, é possível traçar uma linha reta que separa os pontos correspondentes a "AND 1" dos pontos correspondentes a "AND 0", tornando o problema linearmente separável.

No entanto, o problema da função lógica "XOR" (OU exclusivo) demonstra um cenário onde os dados não são linearmente separáveis. Quando os pontos correspondentes a "XOR 1" e "XOR 0" são plotados, não é possível traçar uma única linha reta que separe as duas classes. Isso ocorre porque o "XOR" é uma função não linear e não pode ser resolvido por um único perceptron.

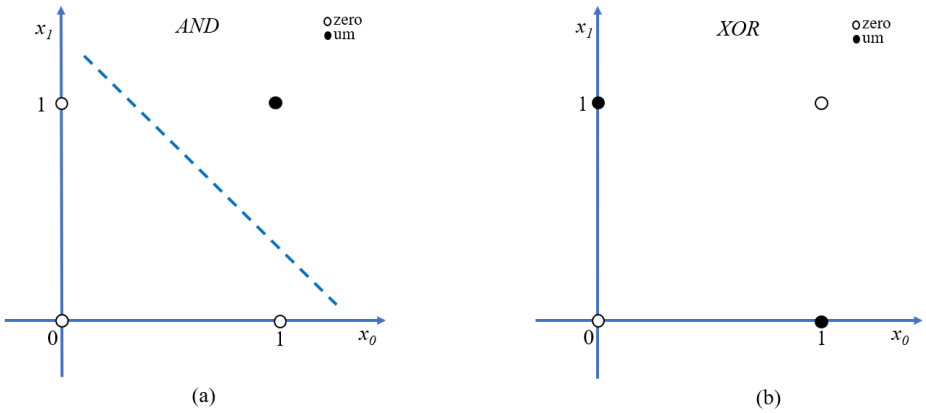


Figura 2.16 – Separabilidade linear em perceptrons.

Para problemas não linearmente separáveis, são necessárias múltiplas camadas de *perceptrons* (*Multilayer perceptron* - MLP). A propagação dos padrões por várias camadas permite a construção de superfícies de decisão que combinam múltiplas superfícies lineares.

O algoritmo utilizado para treinar as MLP é o *Backpropagation*, proposto por Rumelhart em 1986 [36]. Esse algoritmo, geralmente chamado de *regra generalizada delta para o aprendizado por retropropagação*, fornece um método de treinamento eficaz. Basicamente, a aprendizagem consiste de dois passos através das diferentes camadas da rede: um passo para frente, a *propagação*, e um passo para trás, a *retropropagação*. No passo para frente, o vetor de entrada é aplicado à entrada da rede, e seus valores se propagam através da rede. Durante o processo de propagação, os pesos sinápticos da rede são todos fixos. No passo para trás, os pesos sinápticos são todos ajustados de acordo com a regra de correção de erro. Especificamente, a resposta real da rede é subtraída de uma resposta desejada para produzir um sinal de erro. Este sinal de erro é então propagado para trás através da rede, e os pesos sinápticos são ajustados para fazer com que a resposta real da rede se aproxime da resposta desejada [23] [37].

2.3.1 Funções de ativação

A saída de um neurônio é calculada pela função de ativação a partir do somatório da Equação 2.5. As funções apresentadas a seguir não apresentam a limitação abrupta utilizada no *perceptron* de Rosenblatt.

- Função logística: Esta forma de não-linearidade (Figura 2.17) sigmoide na sua forma geral é definida por $y = \frac{1}{1+\exp(-s)}$.

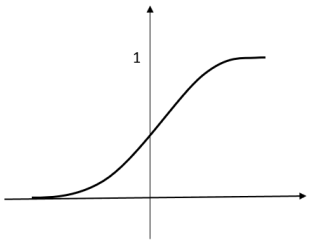


Figura 2.17 – Função logística.

- Retificador linear (*Rectified Linear Unit* - ReLU): é uma função de retificação, ou seja, dados os valores de entrada, os valores de saída correspondem a 0 ou a um valor positivo. O seu processamento, em geral, costuma ser rápido por não envolver cálculos exponenciais, obtendo bons resultados em processamento de imagens [32] [37] [38] . Esta função é definida por: $y = \max(0, s)$, onde s é a soma pondera de todas as entradas acrescentada do bias. A saída dessa função é ilustrada na Figura 2.18. O cálculo dessa função, além de ser rápido, se opõe as outras funções, como por exemplo a logística que satura em 1 para valores altos de entrada. Essa função é amplamente utilizada para projetar redes neurais atualmente [3] [6] [32] [39].

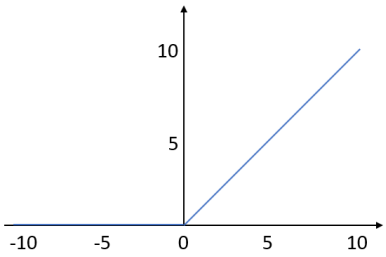


Figura 2.18 – Função ReLU.

- *Softmax*: A função *softmax* é utilizada na última camada da rede neural onde o número de neurônios é igual à quantidade de classes que o problema possui [32] . A saída de cada neurônio compõe um vetor que será então normalizado pela função *softmax*. O resultado é a probabilidade, entre $[0, 1]$, do vetor de entrada pertencer a cada uma das classes possíveis. Ou seja, num problema com 3 classes, por exemplo, a função *softmax* vai produzir 3 valores, que somam 1, onde cada valor representa a probabilidade de a instância pertencer a uma das 3 possíveis classes.

2.4 Visão computacional

Em 1982, a obra "*Computer Vision*" de Ballard e Brown [40] definiu a Visão Computacional como a ciência que estuda e desenvolve tecnologias que permitem que máquinas enxerguem e extraiam características do ambiente por meio de imagens capturadas por diferentes tipos de sensores e dispositivos. Essas informações extraídas permitem o reconhecimento, manipulação e processamento de dados sobre os objetos que compõem a imagem capturada. Embora trabalhos como o de Ballard e Brown tenham sido publicados nos anos 80, as primeiras menções sobre visão computacional ocorreram na década de 50, com os primeiros trabalhos sendo iniciados cerca de 20 anos depois.

Os sistemas de visão computacional estão cada vez mais presentes em nossas vidas. Um exemplo notável são os veículos autônomos que podem trafegar sem intervenção humana. Um sistema de visão computacional é capaz de detectar placas, ruas, pessoas, sinalizações e outros objetos. Além disso, essa tecnologia é amplamente utilizada na medicina para extrair informações e gerar diagnósticos a partir de imagens de ressonâncias, radiografias, etc. A indústria utiliza esses métodos para análise de qualidade de produtos, enquanto a agronomia aplica a visão computacional na irrigação e na aplicação de defensivos [32] [41]. Neste trabalho, a visão computacional será aplicada para reconhecer os alimentos presentes em uma refeição. Um sistema de visão computacional tradicional pode ser dividido nas seguintes etapas [42]:

- **Aquisição**: nesta etapa, as imagens são capturadas.
- **Processamento da imagem**: o objetivo é ajustar e otimizar as imagens, aplicando técnicas como remoção de ruído, rotação da imagem e filtros.
- **Segmentação**: envolve a seleção de regiões de interesse específicas que contêm objetos de interesse.
- **Extração de características**: características matemáticas da imagem são extraídas em vários níveis de complexidade, como detecção de bordas e cantos.
- **Reconhecimento de Padrões**: as imagens são classificadas ou agrupadas de acordo com seu conjunto de características.

O processo de visão computacional é estruturado de maneira modular, respeitando as etapas mencionadas anteriormente, e cada etapa pode fazer uso de diferentes tecnologias disponíveis.

Inicialmente, uma das tecnologias usadas na etapa de reconhecimento de padrões em visão computacional foram as redes MLP totalmente conectadas. Por exemplo, na Figura 2.19 é apresentada uma arquitetura de rede MLP com um vetor de entrada correspondente a uma imagem pequena de dimensões 5×4 pixels, representando a letra "O". O vetor de entrada dessa rede possui 20 valores, onde cada valor está conectado a todos os neurônios da camada de

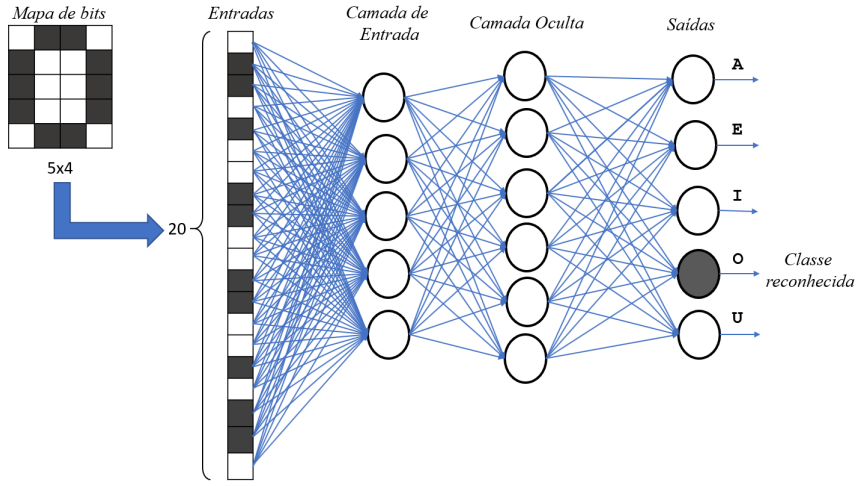


Figura 2.19 – Exemplo de uma MLP e suas conexões.

entrada. Os demais neurônios também são totalmente conectados, e cada conexão representa um parâmetro da rede, cujo valor é definido e ajustado durante o treinamento.

Entretanto, as redes totalmente conectadas apresentam limitações ao lidar com imagens maiores devido ao grande número de parâmetros necessários. Por exemplo, em uma imagem de 256×256 pixels, o vetor de entrada teria 65536 valores. Se a primeira camada da rede tiver 400 neurônios, isso resultaria em 26.214.400 parâmetros apenas na primeira camada. O número total de parâmetros em todas as camadas torna o modelo ineficiente.

As Redes Neurais Convolucionais (*Convolutional Neural Network* - CNN) solucionam esse problema ao utilizar camadas parcialmente conectadas, onde os pixels de entrada não estão conectados a todos os neurônios. As características das imagens são obtidas por meio de camadas de convolução e a dimensionalidade é reduzida em camadas de *pooling* [32]. O resultado desse pré-processamento da imagem é uma representação com baixa dimensionalidade que pode ser submetida a uma rede MLP para a classificação final. As técnicas utilizadas por essas redes, como *convolução* e *pooling*, serão apresentadas nas Seções 2.5.1 e 2.5.2.

As CNNs introduziram um novo conjunto de possibilidades na área de visão computacional. Enquanto no início deste tópico foram mencionadas as etapas de um sistema de visão computacional modular, que utilizavam redes MLP juntamente com processos específicos de pré-processamento de imagem, segmentação e extração de características, essa estrutura modular não é mais necessária quando se utiliza uma CNN na implementação de um sistema de visão computacional capaz de processar, segmentar, extrair características e classificar imagens [42].

2.5 Redes Neurais Convolucionais

Como mencionado na seção anterior, a visão computacional é uma tarefa desafiadora. A solução encontrada para esse problema resultou de extensas pesquisas realizadas por diversos cientistas, que se dedicaram a aplicar CNNs em tarefas práticas de visão computacional [32]. Um marco significativo nessa área foi um artigo publicado em 1998 por Yann LeCun, Léon

Bottou, Yoshua Bengio e Patrick Haffner, que apresentou a arquitetura LeNet-5, amplamente utilizada para o reconhecimento de números em cheques manuscritos [31]. Essa arquitetura consiste em uma rede neural tradicional com funções de ativação sigmoide, bem como camadas de convolução e *pooling*.

O desempenho dos sistemas de visão computacional baseados em CNNs levou empresas de tecnologia, como Google, Facebook, Microsoft, IBM, Twitter e Adobe, bem como um número crescente de *startups*, a embarcarem em projetos de pesquisa e desenvolvimento de produtos e serviços relacionados ao processamento de imagens [37].

A Figura 2.20 apresenta a estrutura fundamental de uma CNN. Inicialmente, a imagem é submetida a filtros de convolução (*kernels*). Cada filtro produz um mapa de características (*feature map*) por meio da operação de convolução. Posteriormente, na camada de *pooling*, cada mapa de características passa por uma operação de *subamostragem* (dimensionalidade reduzida) para diminuir a complexidade, visando reduzir o número de parâmetros [32] [31]. Por fim, os valores resultantes do *pooling* são organizados em um vetor por meio do processo de *flattening*. Esse vetor serve como entrada para a última camada, que normalmente consiste em uma rede neural totalmente conectada, encarregada de classificar a representação pré-processada da imagem gerada pelas camadas convolucionais.

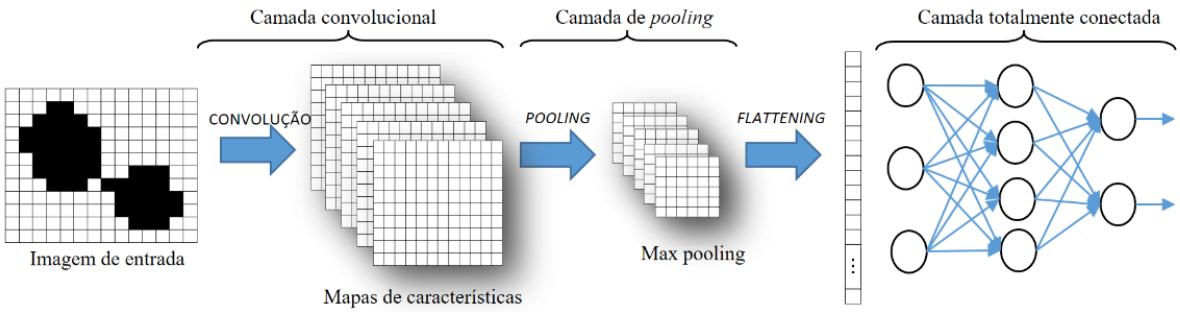


Figura 2.20 – Arquitetura básica de uma CNN.

As arquiteturas das CNNs mais recentes podem conter de 10 a 20 camadas, centenas de milhões de pesos e bilhões de conexões entre unidades. Enquanto há uma década, o treinamento dessas redes complexas poderia levar semanas, o progresso na paralelização de hardware e software reduziu o tempo de treinamento para apenas algumas horas [37].

A seguir, serão discutidas as operações específicas realizadas nas camadas de convolução, *pooling* e o processo de *flattening* em uma CNN.

2.5.1 Camada de convolução

Neste tópico será abordado o processo de convolução, cujo objetivo é extrair características das imagens por meio de filtros. A Figura 2.21 apresenta uma representação matricial simplificada de uma imagem 7×7 e um filtro 3×3, onde, por questões de simplicidade, são utilizados valores binários (0 e 1) para os filtros. Em situações reais, os filtros podem conter valores quaisquer, incluindo negativos.

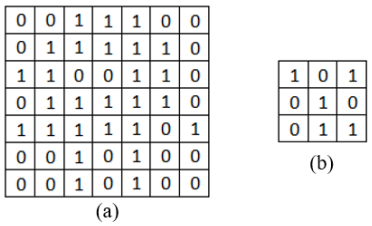


Figura 2.21 – Convolução. (a) Imagem. (b) Filtro de convolução.

Na Seção 2.2.5, foram discutidos alguns filtros que realçam características ou suavizam imagens. As camadas convolucionais em redes neurais operam de maneira semelhante, com a diferença de que seus valores são definidos pelo algoritmo de treinamento, como o *backpropagation*, para auxiliar na classificação das entradas, minimizando o erro de saída [37] [32] [41] [42].

O erro de saída da rede é uma única medida que quantifica o quão distante a resposta da rede está da resposta desejada, permitindo ao algoritmo avaliar o desempenho durante o treinamento. O objetivo é encontrar o conjunto de pesos (valores dos filtros) que minimize esse erro. Além do algoritmo de *backpropagation*, existem outros, como o RMSProp [43] e o ADAM [44], sendo que o algoritmo ADAM é amplamente utilizado em redes convolucionais [45]. A escolha do algoritmo de treinamento depende da natureza do problema, uma vez que alguns métodos são mais adequados para determinados tipos de dados [46].

A operação de convolução gera um mapa de características que corresponde à imagem de entrada. Cada posição desse mapa de características corresponde a um neurônio na camada convolucional. Ao contrário das redes totalmente conectadas, onde cada neurônio está ligado a todos os pixels da imagem de entrada, na convolução, os neurônios estão conectados apenas aos pixels abrangidos pelo filtro de convolução.

A Figura 2.22 ilustra o processo de convolução para o primeiro valor do mapa de características, que é calculado como: $(0 \times 1) + (0 \times 0) + (1 \times 1) + (0 \times 0) + (1 \times 1) + (1 \times 0) + (1 \times 0) + (1 \times 1) + (0 \times 1) = 3$. O cálculo desse valor segue o procedimento detalhado na Seção 2.2.5, que envolve mover uma máscara pela imagem e calcular a soma dos produtos em cada posição.

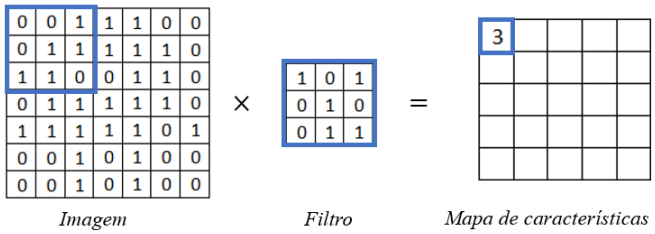


Figura 2.22 – Convolução.

A Figura 2.23 demonstra o deslocamento (*stride*) da janela de convolução e os valores correspondentes no mapa de características. Neste exemplo, o *stride* é definido com o valor um.

Finalmente, na Figura 2.24, é exibido o mapa de características com todos os valores calculados.

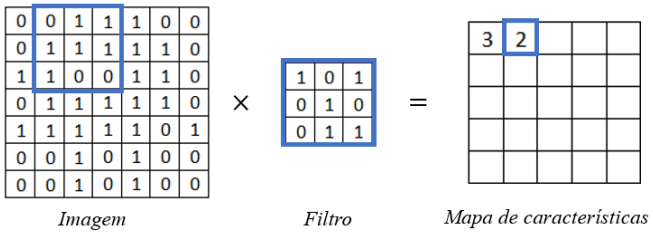


Figura 2.23 – Deslocamento da janela de convolução.

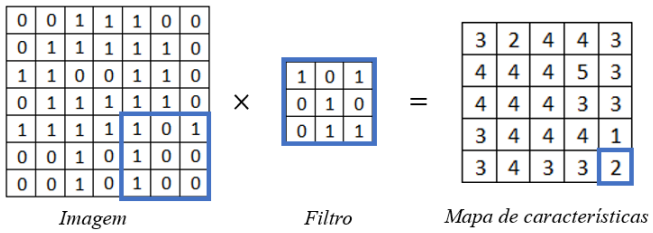


Figura 2.24 – Mapa de características com todos os valores calculados.

O mesmo filtro é compartilhado entre todos os neurônios do mapa de características, resultando em uma redução do número total de parâmetros (pesos) do modelo. Além disso, uma vez que uma CNN aprende a reconhecer um padrão em uma determinada localização da imagem, ela pode reconhecê-lo em qualquer outra parte da imagem, o que difere das redes totalmente conectadas, que aprendem a reconhecer um padrão em uma localização específica [32].

O tamanho do mapa de características também pode ser reduzido modificando o deslocamento (*stride*) da janela de convolução. A Figura 2.25 ilustra a convolução com um *stride* igual a dois, resultando em uma redução na dimensionalidade do mapa de características.

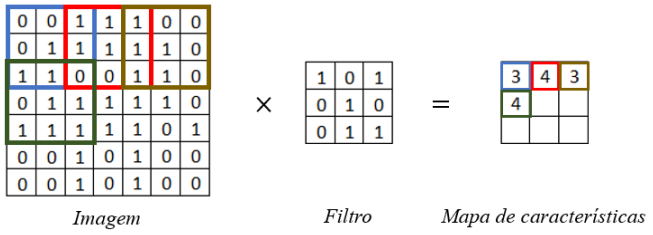


Figura 2.25 – *Stride* igual a 2.

A etapa de convolução é finalizada com a aplicação da função de ativação em cada valor (neurônio) do mapa de características. Uma das funções de ativação mais utilizadas é a função ReLU, que foi apresentada na Seção 2.3.1 e é definida como: $y = \max(0, s)$. Nessa equação, s representa o valor resultante da convolução. A Figura 2.26 ilustra o resultado da aplicação da função de ativação ReLU. Neste exemplo, o mapa de características continha valores negativos.

Nos exemplos apresentados, a convolução foi realizada com apenas um filtro, gerando um único mapa de características. No entanto, como ilustra a Figura 2.20, a camada convolucional pode ter vários mapas de características, cada um obtido através de um filtro específico. Durante o treinamento, a CNN ajusta os parâmetros de cada um desses filtros de forma a representar adequadamente as características da imagem.

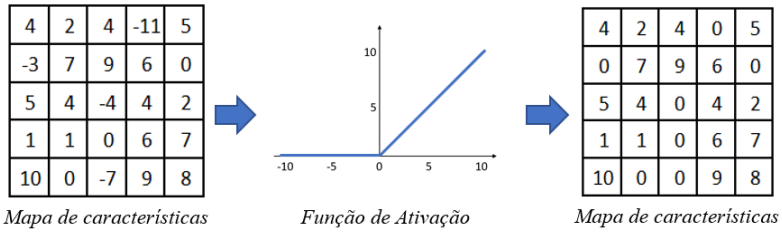


Figura 2.26 – Aplicação da função de ativação.

2.5.2 Camada de *pooling*

A camada de *pooling*, também conhecida como subamostragem, tem como função principal reduzir a dimensão dos dados de entrada. Em uma CNN, os dados de entrada para a camada de *pooling* são os mapas de características gerados na camada de convolução. A função de *pooling* substitui os valores em uma janela por uma estatística, como o máximo, a média ou a média ponderada. A Figura 2.27 exemplifica a operação de máximo usando uma janela 2×2 e um *stride* com valor igual a dois.

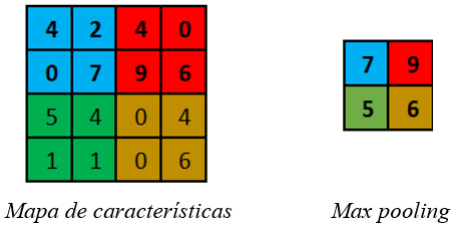


Figura 2.27 – Operação de pooling com janela 2×2 e stride 2.

Além de reduzir a dimensão da representação dos dados, a subamostragem também cria uma invariância a pequenas mudanças e distorções locais, ou seja, permite que as representações variem muito pouco quando os elementos da camada anterior mudam de posição ou aparência [37].

Para um mapa de características com dimensões $w \times h$, as dimensões da saída após a operação de pooling podem ser calculadas como: $\left(\frac{w-f}{s}+1\right) \times \left(\frac{h-f}{s}+1\right)$, onde w e h são as dimensões da imagem, f é o tamanho do filtro de *pooling* e s é o tamanho do *stride*.

2.5.3 *Flattening*

Após todas as etapas de convoluções e *poolings*, os mapas de características gerados serão transferidos para um vetor. Esse vetor contém o resultado do pré-processamento da imagem original, que agora possui uma representação dos pixels com apenas características relevantes para a classificação. Esse vetor é a entrada da rede neural totalmente conectada, que é responsável por classificar a imagem. A Figura 2.20 ilustra a representação desse vetor.

2.6 Redes Completamente Convolucionais

A tarefa de classificação de imagens consiste em atribuir um rótulo ou classe a uma imagem de entrada. No entanto, em algumas situações, é desejável conhecer a localização de objetos na imagem, sua forma ou a que objeto cada pixel pertence. Essa necessidade leva à segmentação de imagens, onde um rótulo é atribuído a cada pixel da imagem. A segmentação de imagens envolve a compreensão da imagem em um nível de pixel e tem aplicações em áreas como imagens médicas, veículos autônomos e análise de imagens de satélite, entre outras.

A Figura 2.28 ilustra um exemplo de segmentação semelhante ao problema abordado neste trabalho. Cada alimento da refeição foi rotulado com uma cor diferente.

Figura 2.28 – Segmentação. (a) Alimentos. (b) Alimentos rotulados.

Para a tarefa de segmentação de imagens usando CNN, os pesquisadores Jonathan Long, Evan Shelhamer e Trevor Darrell introduziram em 2015 a rede completamente convolucional (*Fully Convolutional Network* - FCN) [47]. A Figura 2.29 mostra a representação dessa arquitetura de rede, que difere de uma CNN convencional por não possuir uma camada de rede neural totalmente conectada na última camada. Em vez disso, a FCN interpreta a imagem de entrada e produz como saída outra imagem. A camada totalmente conectada é substituída por uma camada de convolução que classifica cada pixel da imagem em uma classe específica. Esse tipo de segmentação é conhecido como segmentação semântica.

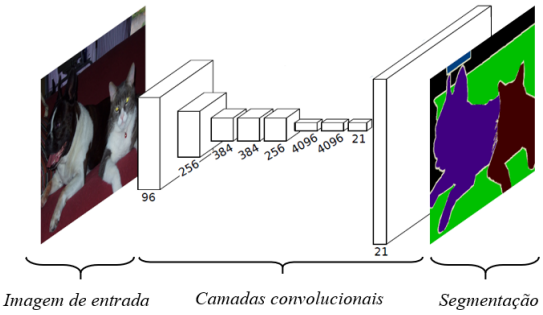


Figura 2.29 – Representação da arquitetura da rede FCN [47].

2.6.1 U-Net

A rede neural U-Net, apresentada por Ronneberger, Fischer e Brox em 2015 [48], foi desenvolvida originalmente para a segmentação de imagens médicas. Essa arquitetura consiste apenas em camadas de convolução. Ela segue uma estrutura conhecida como "*encoder-decoder*", composta por camadas de contração (*downsampling*) à esquerda da rede e camadas de expansão (*upsampling*) à direita, criando uma simetria que se assemelha à letra "U".

A Figura 2.30 ilustra a arquitetura da rede U-Net, onde cada caixa azul representa um mapa de características, com o número de mapas indicado acima de cada caixa. No canto inferior esquerdo, a dimensão da imagem é especificada. As caixas brancas representam cópias dos mapas de características, e cada flecha com uma cor específica representa uma operação distinta, conforme explicado no canto inferior direito da figura. No lado esquerdo da rede, as flechas vermelhas correspondem ao caminho de contração, enquanto no lado direito da rede, as flechas verdes se referem ao caminho de expansão. Essa operação é usada para restaurar a dimensão da imagem ao seu estado anterior à operação de convolução. Na camada final, uma convolução com *kernel* 1×1, juntamente com uma função de ativação *softmax*, é empregada para realizar a classificação pixel a pixel da imagem de saída.

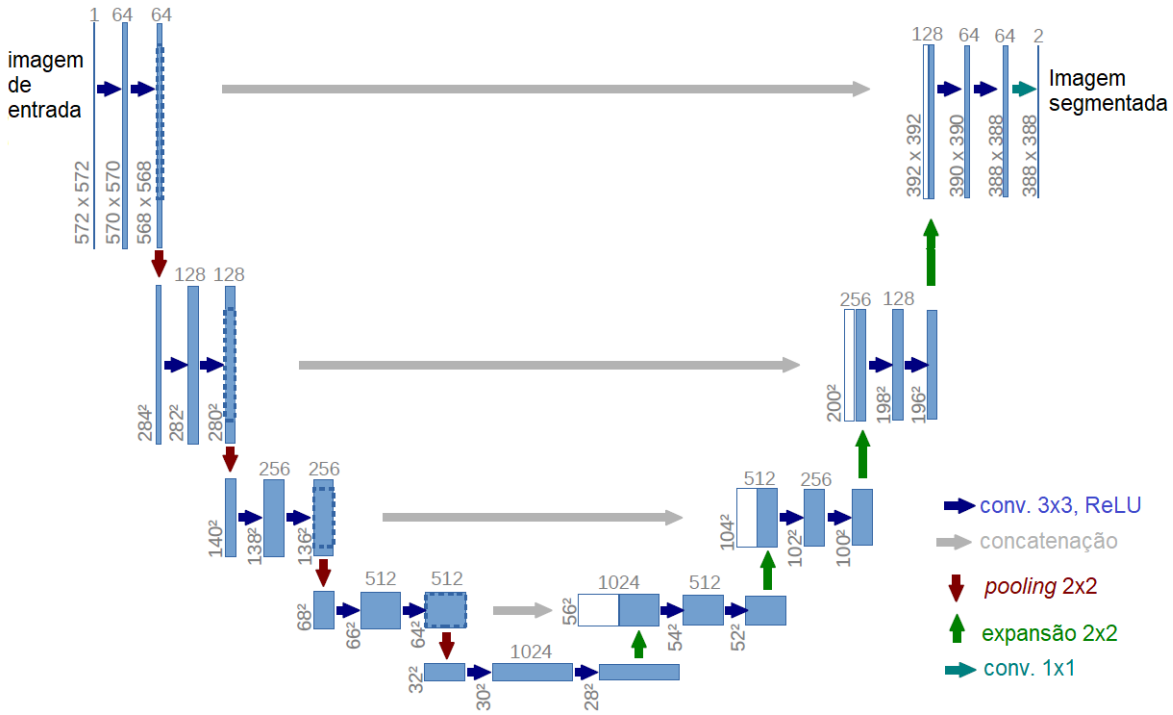


Figura 2.30 – Representação esquemática da arquitetura da rede U-Net proposta por Ronneberger, Fischer e Brox [48].

2.6.2 SegNet

A SegNet é outra arquitetura de rede FCN usada para realizar a segmentação semântica de imagens. Foi proposta por Badrinarayanan *et al.* [49] e, assim como a U-Net, divide-se em

duas partes principais: o *codificador* e o *decodificador*. O codificador é responsável por extrair características da imagem de entrada, enquanto o decodificador reconstrói a imagem segmentada com base nessas características [49]. A Figura 2.31 apresenta a arquitetura da SegNet.

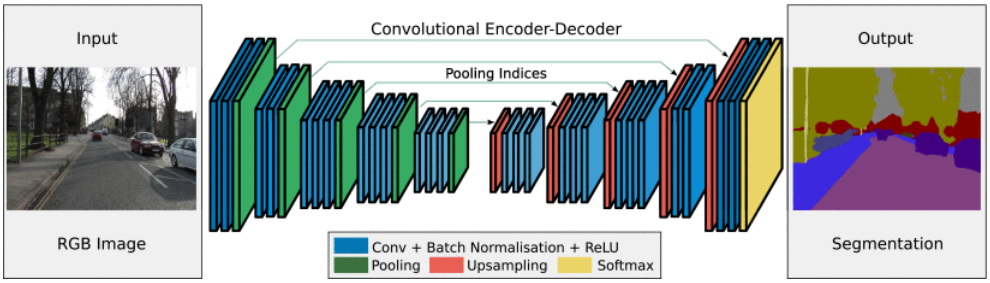


Figura 2.31 – Arquitetura SegNet: codificador e decodificador [49].

Na arquitetura da SegNet, o codificador realiza convoluções para gerar mapas de características, que são normalizados antes da aplicação da função ReLU. Em seguida, é realizada a operação de *pooling* usando uma janela de tamanho 2×2 e um passo (*stride*) com valor igual a 2, reduzindo pela metade as dimensões dos mapas de características. Ao final do codificador, esses mapas de características têm baixa resolução, mas contêm informações que descrevem o contexto da imagem de entrada como um todo [49].

A diferença-chave em relação a outras FCNs está no processo de *pooling*, pois a SegNet precisa armazenar informações para uso no decodificador. Os autores propuseram o armazenamento dos índices correspondentes ao *pooling* máximo, ou seja, a posição dos maiores valores de pixels. A Figura 2.32 ilustra o processo de *pooling* e *upsampling*. A camada de *pooling* reduz o mapa de características e gera os índices dos maiores valores em cada janela de *pooling*, que são números inteiros predefinidos para cada posição. A camada de *upsampling* recebe o mapa de características reduzido e os índices para construir um mapa de ativação que aumenta a resolução.

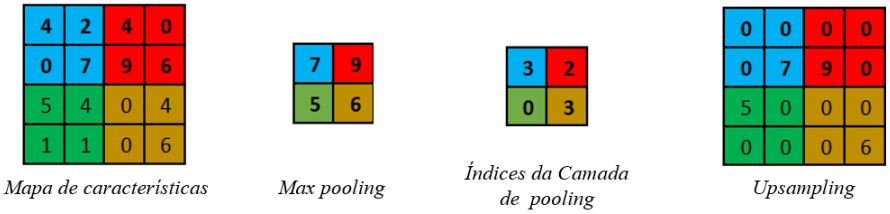


Figura 2.32 – Camada de *pooling* e *upsampling*.

É importante notar que o processo de *upsampling* produz zeros no mapa de ativação. Portanto, os filtros dessa camada têm como objetivo aprender os valores dos pixels zerados, convergindo assim para a segmentação desejada.

2.6.3 MobileNets

A arquitetura de rede neural conhecida como MobileNet foi proposta por Andrew G. Howard *et al.* [50] com foco em aplicações em dispositivos móveis. Essa arquitetura pode ser aplicada a problemas de classificação ou segmentação, sendo capaz de atuar como um codificador, cujas saídas podem ser combinadas com um decodificador adaptado ao problema, permitindo a construção de uma rede neural para segmentação semântica com características robustas e um número reduzido de parâmetros treináveis [51] [52].

A ideia central dessa arquitetura é substituir as camadas convolucionais, que são computacionalmente intensivas, por convoluções separáveis em profundidade. Nesse modelo, a camada de convolução é dividida em subtarefas que, quando combinadas, formam um bloco de convolução separável em profundidade, reduzindo significativamente o custo computacional em até nove vezes [53].

Como mencionado anteriormente, a aplicação de filtros, funções de ativação ou *pooling* gera mapas de características. Na MobileNet, esses mapas de características são referidos como canais de uma imagem.

Em uma convolução convencional, quando a imagem de entrada tem, por exemplo, três canais (como os canais RGB), a aplicação de um único filtro de convolução resulta em uma imagem de saída com apenas um canal por pixel. A Figura 2.33 ilustra esse processo.

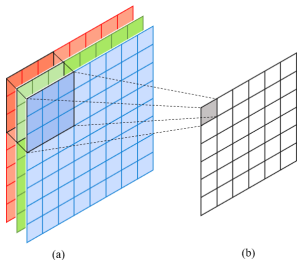


Figura 2.33 – Convolução convencional. (a) Imagem de entrada com três canais (RGB). (b) Cada janela de convolução gera apenas um pixel [53].

Ao contrário da convolução convencional, a convolução separável em profundidade não combina os canais de entrada, mas realiza a convolução em cada canal separadamente. Para uma imagem com três canais de entrada, uma convolução em profundidade cria uma imagem de saída que também possui três canais. Cada canal recebe seu próprio conjunto de pesos. A Figura 2.34 ilustra esse tipo de convolução.

A convolução em profundidade é seguida por uma convolução pontual, que é equivalente a uma convolução convencional, mas com um filtro de tamanho 1×1. Isso permite combinar os pixels de cada canal de entrada em um único canal de saída. A Figura 2.35 ilustra esse processo.

O objetivo da convolução em profundidade é filtrar os canais de entrada para realçar características relevantes, como detecção de bordas, linhas, filtragem de cores, entre outras. A capacidade de filtragem de cores é particularmente importante na segmentação semântica, onde

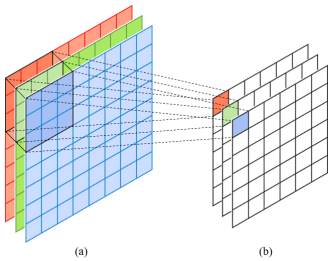


Figura 2.34 – Convolução em profundidade. (a) Imagem com três canais de entrada. (b) A convolução em profundidade cria três canais de saída [54].

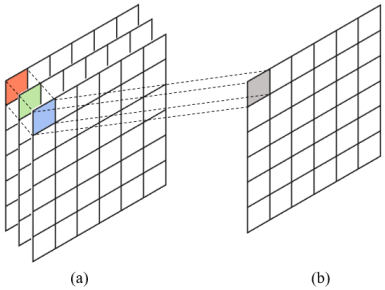


Figura 2.35 – Convolução pontual. (a) Canais gerados pela convolução profunda. (b) Convolução pontual com filtro 1×1 combinando os pixels de cada canal em um único pixel [54].

as informações de cor desempenham um papel significativo na classificação de pixels em suas respectivas classes.

A MobileNet pode ser utilizada para construir um modelo de uma U-Net modificada para segmentação semântica sendo utilizada como codificador na execução do *downsampling* e suas saídas combinadas com um decodificador adaptado ao problema [54].

Os tópicos a seguir descrevem algumas das versões dessa arquitetura apresentados pelos autores.

2.6.3.1 MobileNet - versão 1

Na primeira versão da arquitetura MobileNet, conhecida como MobileNetV1, as operações de convolução em profundidade e convolução pontual são combinadas em um único *bloco de convolução em profundidade*. Cada bloco é seguido por uma normalização em lote e pela aplicação da função de ativação ReLU6 [53]. A arquitetura completa do MobileNetV1 começa com uma camada convencional de 3×3 como a primeira camada, seguida por 13 repetições do bloco de construção mostrados na Figura 2.36.

Não há camadas de *pooling* entre os blocos separáveis em profundidade. Em vez disso, algumas das camadas de profundidade têm um *stride* igual a dois para reduzir as dimensões espaciais dos dados [53]. Após as camadas de convolução, a normalização em lote e a aplicação da função ReLU6 são aplicadas para evitar ativações com valores muito grandes. A função ReLU6 é semelhante à ReLU, mas com um limite superior de seis para suas saídas, definido na equação $y = \min(\max(0, x), 6)$. Os autores relatam em [53] que a função ReLU6 se mostrou

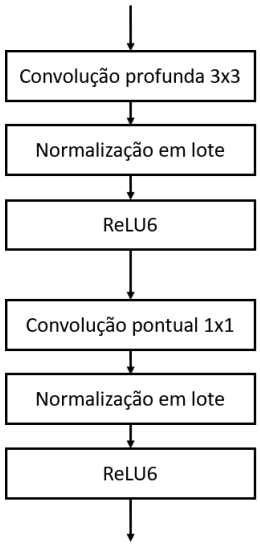


Figura 2.36 – Bloco de convolução em profundidade e camadas pontuais seguidas por normalização em lote e ReLU6 [53].

mais robusta do que a ReLU. A saída dessa função é ilustrada na Figura 2.37.

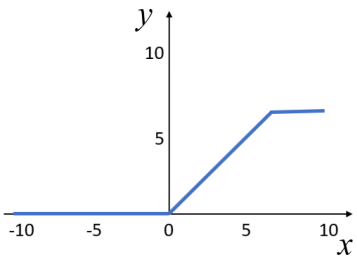


Figura 2.37 – Função de ativação ReLU6 [53].

2.6.3.2 MobileNet - versão 2

A MobileNetV2, apresentada em [53], representa um refinamento em relação à versão anterior, visando melhorar o desempenho. O bloco de construção principal da MobileNetV2 é chamado de "bloco residual de gargalo" (*bottleneck residual block*), como mostrado na Figura 2.38.

Este bloco de construção consiste em três camadas convolucionais. A primeira camada é uma convolução 1×1 que tem como objetivo expandir o número de canais antes de aplicar a segunda camada. A segunda camada é uma convolução em profundidade, enquanto a terceira camada, chamada de camada de projeção, tem a finalidade de reduzir o número de canais de volta ao tamanho original. O *fator de expansão* que determina o tamanho da expansão é um parâmetro que pode ser definido, sendo o valor padrão igual a seis [55].

A Figura 2.39 ilustra o processo de expansão e projeção usando um exemplo de entrada com 24 canais e um fator de expansão com valor igual a seis, resultando em 144 canais após a primeira convolução e, em seguida, projetando-os de volta para 24 canais.

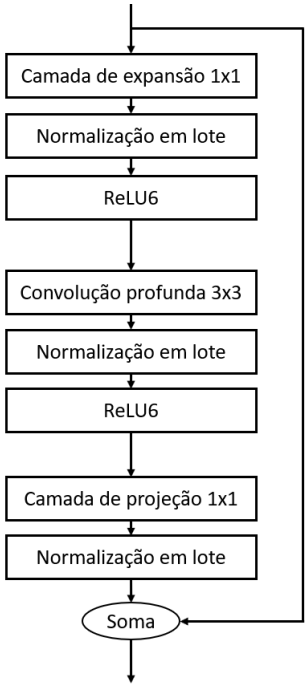


Figura 2.38 – *Bottleneck residual block* [53].

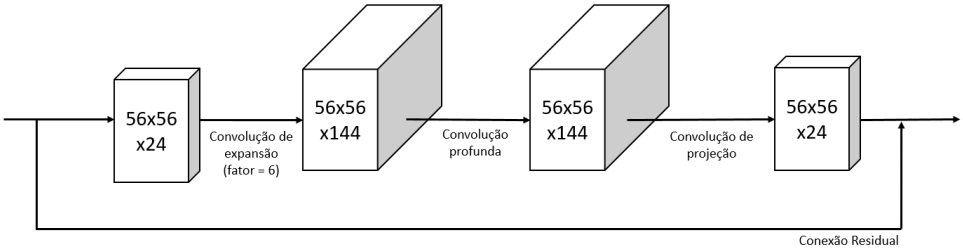


Figura 2.39 – Expansão e projeção [53].

Cada camada de convolução é seguida por normalização em lote e pela aplicação da função de ativação ReLU6. No entanto, a saída da camada de projeção não tem uma função de ativação aplicada a ela. Os autores relatam que o uso de uma não linearidade após essa camada resulta em perda de informações úteis [53].

A arquitetura completa do MobileNetV2 consiste em 17 desses blocos de construção. Esses blocos são seguidos por uma convolução convencional 1×1, uma camada de *pooling* médio e, finalmente, uma camada de classificação [53].

2.7 Métricas de avaliação

As métricas de avaliação desempenham um papel fundamental na avaliação do desempenho de modelos de classificação. Essas métricas são calculadas com base na classificação de um conjunto de dados de teste que não foi usado no treinamento do modelo. Para calcular essas métricas e compreender o desempenho do modelo, a matriz de confusão é uma ferramenta essencial.

O conjunto de treinamento é geralmente dividido em dois conjuntos distintos, um para treinamento e outro para teste do modelo. O conjunto de teste normalmente consiste em uma fração do total de exemplos disponíveis e deve conter exemplos de todas as classes utilizadas no treinamento. Após o treinamento, esses exemplos são submetidos ao modelo para obter medidas de classificações, que podem ser corretas ou incorretas. No entanto, em um problema de classificação binária existem duas outras classes possíveis, denominadas classes positiva e negativa. As informações correspondentes aos resultados dos testes podem ser representadas por meio da **matriz de confusão**.

A matriz de confusão é uma ferramenta essencial para avaliar o desempenho de um modelo de classificação. Ela é derivada das medidas obtidas nos testes e fornece uma tabela que compara as classificações corretas com as classificações previstas para cada classe em um conjunto de exemplos. Basicamente, a matriz de confusão indica os erros e acertos do modelo em comparação com os resultados esperados [56]. Para cada classe, são extraídas quatro variáveis:

Verdadeiro positivo (VP): quando a entrada é classificada como positiva e realmente pertence a essa classe.

Verdadeiro negativo (VN): quando a entrada é classificada como negativa e realmente não pertence a essa classe.

Falso positivo (FP): quando a entrada é classificada como positiva, mas na verdade não pertence a essa classe.

Falso negativo (FN): quando a entrada é classificada como negativa, mas na verdade pertence a essa classe.

A Figura 2.40 apresenta um exemplo de uma matriz de confusão.

		Classe detectada	
		Sim	Não
Classe original	Sim	VP	FN
	Não	FP	VN

Figura 2.40 – Matriz de Confusão .

A partir dos valores contidos na matriz de confusão, é possível calcular várias métricas de avaliação para a classificação. Abaixo, são apresentadas algumas dessas métricas:

A **acurácia** avalia a porcentagem de acertos, sendo obtida pela razão entre a quantidade de acertos e o total de entradas. Sua fórmula é dada por:

$$\text{Acurácia} = \frac{VP + VN}{VP + FN + VN + FP} \tag{2.6}$$

A **sensibilidade** avalia a capacidade do modelo em detectar com sucesso resultados classificados como positivos. É útil em situações onde os falsos negativos são considerados mais prejudiciais que os falsos positivos e é calculada conforme:

$$\text{Sensibilidade} = \frac{VP}{VP + FN} \tag{2.7}$$

A **precisão** é usada quando os falsos positivos são considerados mais prejudiciais que os falsos negativos, sendo calculada pela fórmula:

$$\text{Precisão} = \frac{VP}{VP + FP} \tag{2.8}$$

Por fim, o **F1-score** é uma métrica que combina sensibilidade e precisão em uma única medida, sendo uma média harmônica que favorece resultados mais próximos ao valor mínimo das métricas individuais. Quando o F1-score é baixo, indica que tanto a sensibilidade quanto a precisão estão em níveis baixos. Sua fórmula é dada por:

$$\text{F1} = 2 \cdot \frac{\text{precisão} \cdot \text{sensibilidade}}{\text{precisão} + \text{sensibilidade}} \tag{2.9}$$

Cada métrica possui características específicas que devem ser consideradas ao escolher como avaliar o desempenho de um modelo de classificação, dependendo do contexto e dos requisitos do problema em questão. No contexto deste trabalho foram registrados os valores de acurácia e sensibilidade obtidos. A sensibilidade foi calculada, uma vez que essa métrica ganha importância em situações em que os Falsos Negativos são considerados mais prejudiciais do que os Falsos Positivos. No contexto deste trabalho, Falso Negativo significa não identificar um alimento presente na refeição, o que pode ser prejudicial na geração de informações sobre a refeição.

2.8 Considerações finais

Neste capítulo, foram apresentados os fundamentos teóricos que serviram de base para o desenvolvimento do modelo proposto e implementado neste trabalho.

O próximo capítulo consiste na apresentação e discussão detalhada do modelo implementado.

3 Modelo proposto

Neste capítulo, será apresentado o modelo proposto para a identificação de alimentos brasileiros em imagens de refeições.

Inicialmente, o problema consistia em segmentar os alimentos colocados separadamente em um prato e, em seguida, identificar o alimento em cada segmento usando uma CNN. No entanto, durante a pesquisa, ficou claro que a semelhança entre as imagens tornava a segmentação uma tarefa difícil.

Diversas arquiteturas de redes neurais foram testadas, sendo que os melhores resultados foram obtidos usando FCN. Exemplos de segmentações usando esse modelo de rede estão ilustrados na Figura 3.1. É possível observar a incerteza na previsão dos rótulos para a segmentação de alimentos. Essas incertezas aumentam à medida que o número de classes de alimentos também aumenta. Como resultado, ocorrem identificações incorretas de alimentos quando eles recebem rótulos incorretos durante a segmentação.

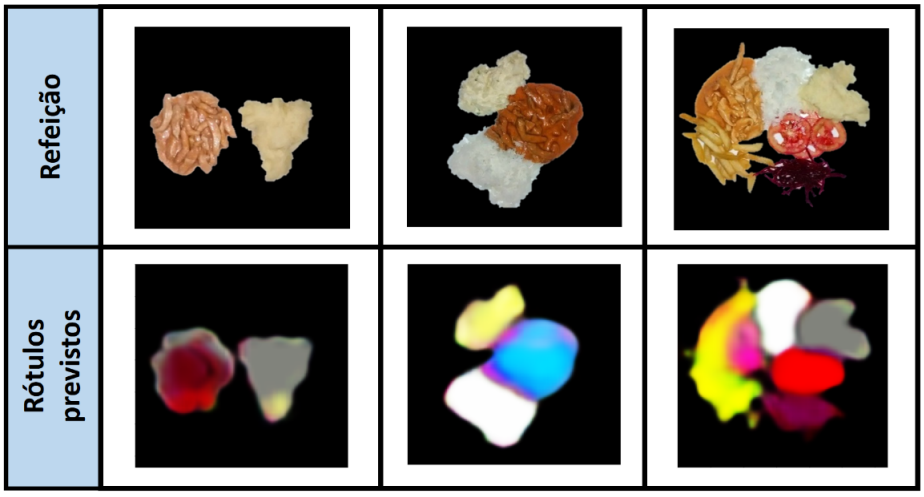


Figura 3.1 – Segmentações obtidas para alimentos usando FCN.

A abordagem proposta nesta pesquisa objetivou melhorar a precisão e fornecer segmentações individuais para cada alimento na refeição.

O modelo proposto é composto por um conjunto de redes completamente convolucionais. Inicialmente, a imagem de uma refeição é submetida ao conjunto de FCNs. As redes desse conjunto que correspondem aos alimentos presentes na refeição geram, cada uma, uma segmentação contendo a localização daquele alimento na refeição. Nesta abordagem, as redes convolucionais não são responsáveis por classificar os alimentos existentes, apenas fornecem um conjunto de segmentações para uma refeição. A classificação dos alimentos é então realizada com base na análise das segmentações produzidas. Para essa análise foi desenvolvido um algoritmo para processar as segmentações e identificar os alimentos presentes na refeição.

A Figura 3.2 mostra a estrutura geral do modelo proposto. A imagem de uma refeição é submetida ao conjunto de FCNs. Cada FCN produz uma imagem correspondente a uma máscara de segmentação para um alimento específico. No exemplo mostrado na figura, a imagem da refeição contém os alimentos "estrogonofe de frango" e "purê de batata". As FCNs correspondentes a esses alimentos geraram suas respectivas segmentações, mas as FCNs para "estrogonofe de carne" e "arroz" também geraram segmentações. Isso ocorre devido à semelhança entre as imagens desses alimentos. A identificação dos alimentos depende da análise de quais segmentações são "verdadeiras", ou seja, aquelas que realmente correspondem a um alimento presente na refeição.

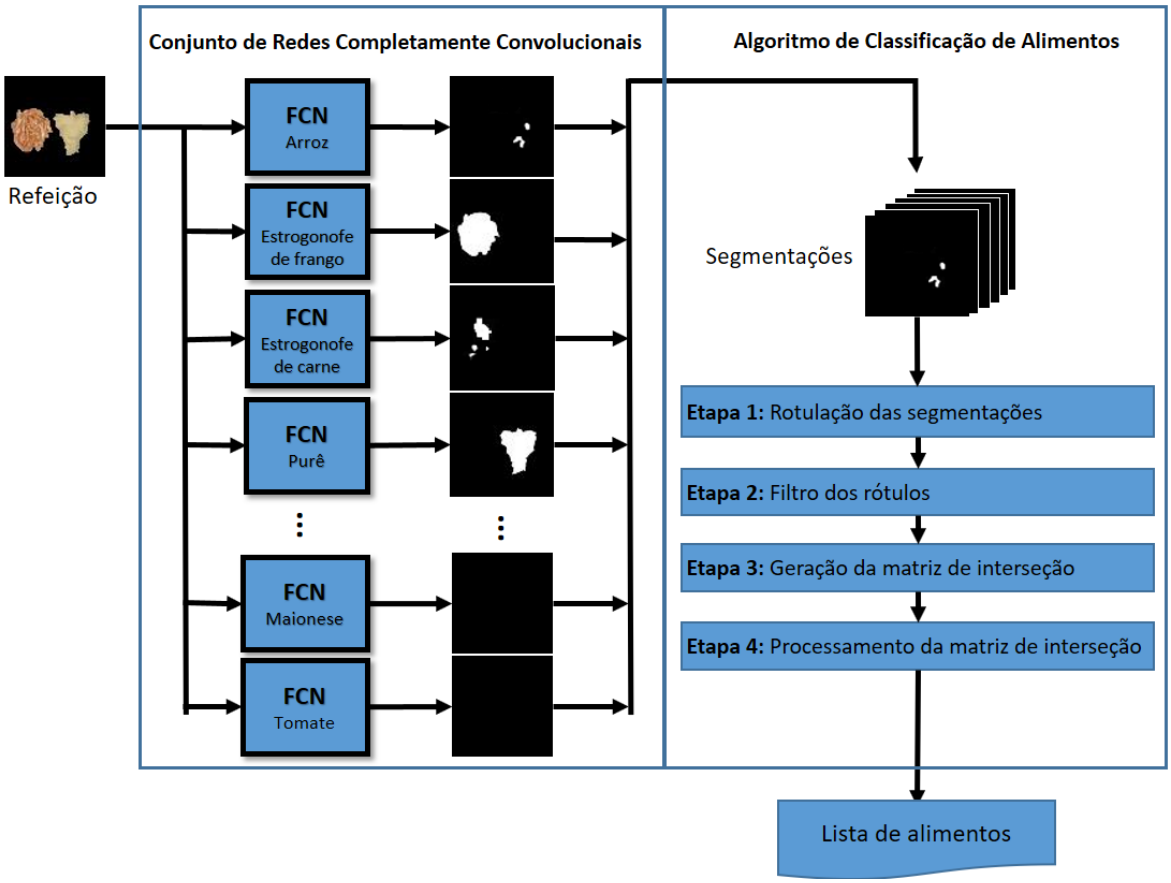


Figura 3.2 – Estrutura geral do modelo proposto.

Para análise das segmentações, foi desenvolvido um algoritmo chamado de Algoritmo de Classificação de Alimentos, que analisa as segmentações, define quais são as mais consistentes e gera a lista dos alimentos da refeição. Conforme apresentado na Figura 3.2, esse algoritmo é composto por quatro etapas e sua implementação utiliza técnicas de processamento de imagens digitais, como rotulagem de componentes conectados, operações lógicas e filtros. As etapas desse algoritmo são apresentadas na Seção 3.4.

3.1 Conjunto de imagens utilizadas para o desenvolvimento do modelo

Para o desenvolvimento do modelo proposto foram utilizadas inicialmente um conjunto com doze alimentos diferentes. Essas imagens foram utilizadas no trabalho apresentado por Shiga [22] e obtidas pelo autor utilizando uma câmera de um telefone celular. O conjunto completo está disponível para download[1].

Os alimentos desse conjunto são:

- alface;
- arroz branco;
- batata frita;
- beterraba;
- cenoura;
- estrogonofe de carne;
- estrogonofe de frango;
- feijão;
- maionese;
- peito de frango à milanesa;
- purê de batatas;
- tomate.

A Figura 3.3 mostra alguns exemplos de imagens do conjunto de treinamento utilizado. Pode-se observar que os alimentos estão dispostos separadamente no prato e as fotos foram feitas com distâncias e em superfícies diferentes.

3.2 Pré-processamento e conjunto de treinamento

No modelo proposto, um conjunto de treinamento específico é construído para cada classe de alimento. Cada conjunto é composto pelas imagens das refeições e suas respectivas máscaras de segmentação. A construção desses conjuntos foi facilitada com o desenvolvimento de uma aplicação em Python. Essa aplicação utiliza o algoritmo *GrabCut* [57], disponível na biblioteca *OpenCV* [58].

[1] https://github.com/yurishiga/bd_porcoes_alimentares

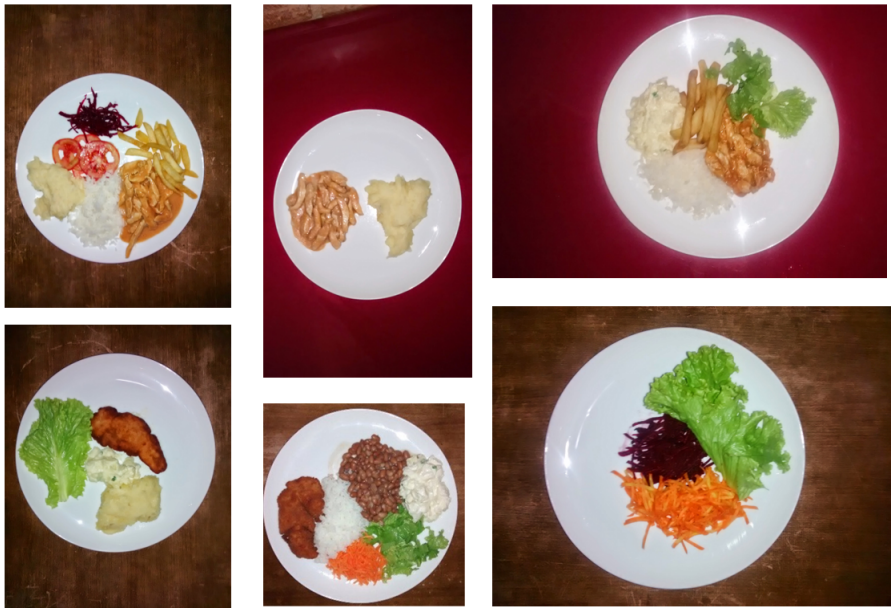


Figura 3.3 – Exemplos de imagens de pratos com alimentos do conjunto de treinamento.

A Figura 3.4 mostra uma das telas dessa aplicação. A Figura 3.5(a) mostra a interface da aplicação sendo utilizada para segmentação do purê de batata. Após o usuário delimitar o alimento na imagem, a aplicação gera automaticamente a máscara de segmentação correspondente, que é então armazenada no conjunto de treinamento específico para aquele alimento. A Figura 3.5(b) exibe a máscara de segmentação resultante desse processo.

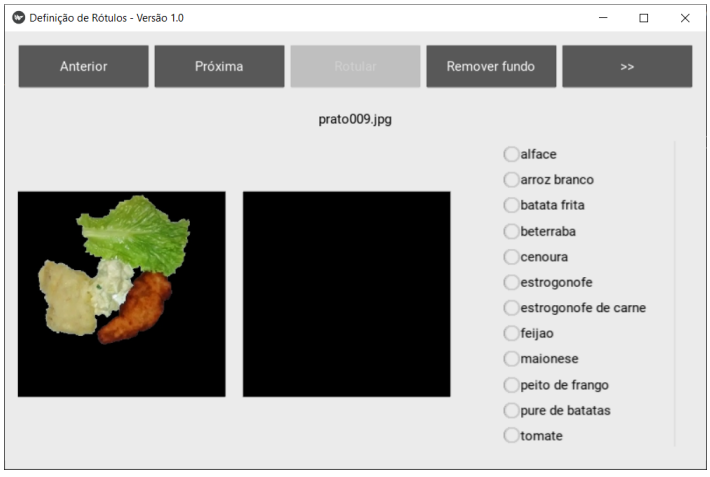
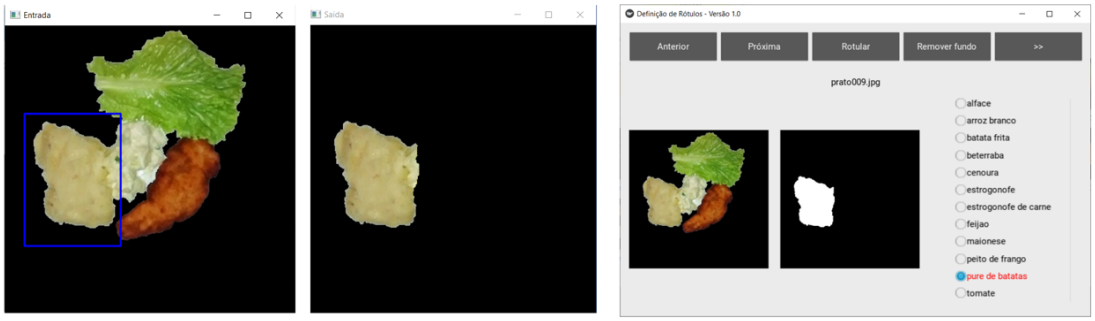


Figura 3.4 – Tela principal da aplicação de rotulação.

A Figura 3.6 apresenta exemplos dessas imagens juntamente com suas segmentações. No exemplo fornecido, os alimentos “Alface”, “Arroz”, “Batata frita”, “Purê” e “Estrogonofe de frango” possuem conjuntos individuais de imagens destinadas ao treinamento do modelo.

Para evitar o *overfitting*, foi realizada um aumento de dados criando novas amostras a partir das existentes. Cada imagem foi replicada oito vezes e cada réplica foi rotacionada em 45 graus. Como discutido na introdução deste trabalho, a restrição de tamanho do conjunto de treinamento foi um dos desafios abordados pelo modelo proposto.



(a) (b)

Figura 3.5 – Segmentação dos alimentos utilizando a aplicação desenvolvida. (a) Seleção do purê de batata. (b) Tela principal mostrando a segmentação correspondente ao alimento selecionado.

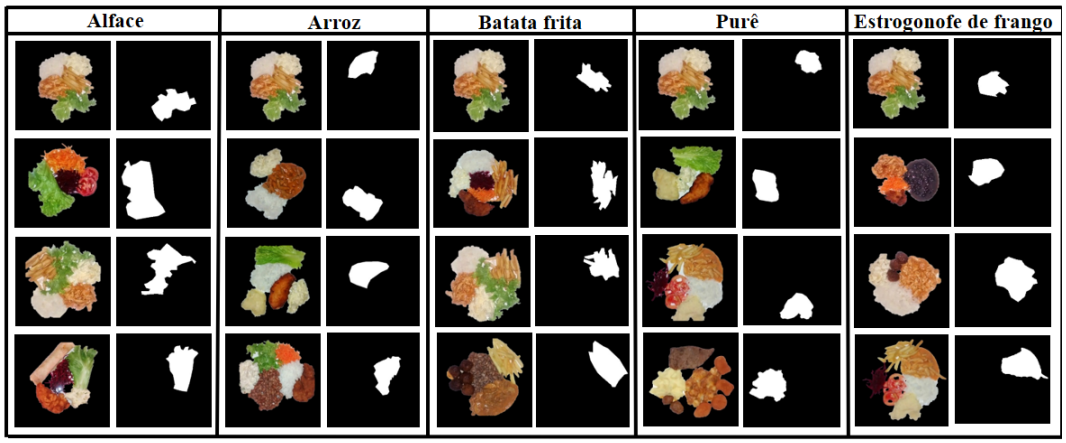


Figura 3.6 – Exemplos de segmentações de alimentos para treinamento do modelo.

Além disso, no que diz respeito à preparação dos dados, as imagens das refeições têm suas bordas recortadas para respeitar os limites do prato e são redimensionadas para 128×128 pixels para se adequarem à entrada do modelo. Por fim, o fundo de todas as imagens foi removido, deixando apenas a imagem dos alimentos. Essas operações foram realizadas por meio da aplicação apresentada nessa seção.

3.3 Arquitetura da FCN

A tarefa de cada FCN na abordagem proposta é atribuir um rótulo a cada pixel na imagem, semelhante à previsão de múltiplas classes, neste caso apenas classes 0 (zero) ou 1 (um). Cada rede aprende a prever uma classe correspondente para um determinado alimento, gerando uma imagem binária correspondente à máscara segmentada. Essa máscara permite conhecer a posição e a forma do alimento. A arquitetura da FCN escolhida para a segmentação é uma U-Net modificada [59]. Uma U-Net consiste em um codificador (*downsampler*) e um decodificador (*upsampler*). O codificador fornece informações sobre as características da imagem de entrada, que incluem bordas, texturas, cores etc. Essas informações são usadas pelo decodificador para gerar a imagem segmentada. Foi utilizado o codificador pré-treinado da arquitetura MobileNetV2 [50] [51][52][53]. Suas saídas intermediárias são usadas pelo decodificador *Pix2Pix* [60],

que é treinado para aprender como mapear a imagem de entrada para a imagem segmentada.

A Figura 3.7 mostra a configuração das arquiteturas MobileNetV2 e *Pix2Pix* usadas para compor as FCNs. Configurações com diferentes números de blocos produziram resultados semelhantes.

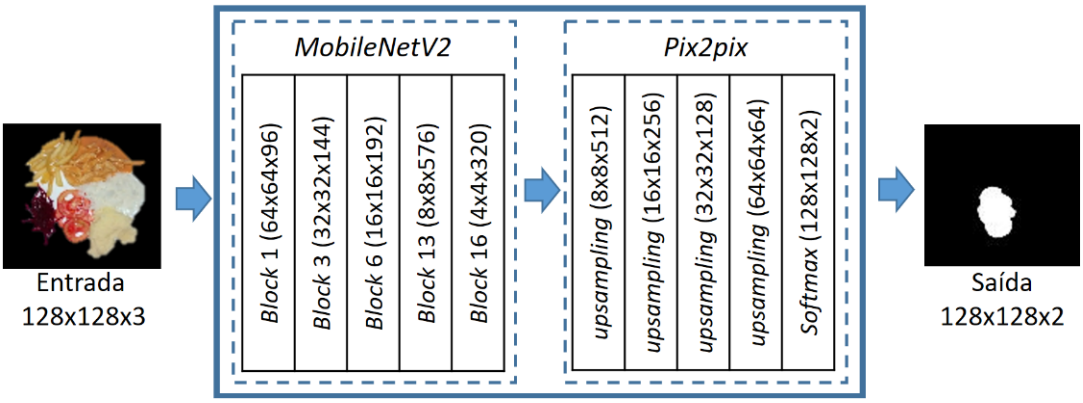


Figura 3.7 – Arquitetura básica das FCN utilizadas no modelo proposto.

A arquitetura MobileNetV2 foi escolhida porque utiliza operações de convolução leves que reduzem o número de operações necessárias para treinar e executar a rede. No entanto, no modelo proposto, também é possível usar outras arquiteturas de rede para gerar segmentações, como ResNet-152V2 [61], InceptionResNetV2 [62], NasNetLarge [63], entre outras. No entanto, todas essas arquiteturas apresentam resultados semelhantes para as classificações mais difíceis [15]. Portanto, o problema de segmentações imprecisas, independentemente da arquitetura usada para a segmentação de imagens, é resolvido nesta abordagem pelo algoritmo de classificação de alimentos que compõe o modelo proposto.

3.4 Algoritmo de Classificação dos Alimentos

O algoritmo de classificação de alimentos desempenha um papel crucial no processo de identificação dos alimentos nas refeições, a partir das segmentações geradas pelas FCNs. Considerando que as segmentações podem ser inconclusivas, esse algoritmo é componente fundamental do modelo proposto.

Cada segmentação produzida corresponde a um alimento específico. A análise dessas segmentações tem como objetivo determinar quais delas são verdadeiras. Como mencionado anteriormente, a semelhança entre os alimentos pode resultar em múltiplas segmentações positivas para um mesmo alimento. Portanto, este algoritmo tem a tarefa de selecionar entre as segmentações positivas qual é a correta ou, se nenhuma for considerada correta, se alguma delas pode ser considerada como uma alternativa de classificação para aquele alimento.

É importante ressaltar que as imagens segmentadas são binárias, com apenas dois valores possíveis para cada pixel: 0 (preto) para segmentação negativa e 1 (branco) para segmentação positiva. Nessas imagens, podem existir ruídos, como discutido na Seção 2.2.6. Para garantir

que esses ruídos não impactem a eficiência do modelo, é aplicado o filtro da mediana em todas as segmentações.

O algoritmo é composto por várias etapas, cada uma gerando informações necessárias para a etapa seguinte, culminando na lista de alimentos que será a saída principal do modelo. A seguir serão apresentadas de forma detalhada cada uma das etapas. A exposição segue uma abordagem descritiva em vez de utilizar estruturas algorítmicas convencionais. A opção por uma descrição passo a passo, visa facilitar a compreensão do processo analítico das segmentações.

Etapa 1: Rotulação das segmentações

Conforme apresentado na Seção 2.2.4.3, a rotulação é um procedimento que tem por objetivo atribuir um rótulo a cada objeto (ou componente conexo) da imagem. O algoritmo de rotulação transforma uma imagem binária em uma representação simbólica que representa os objetos conectados no qual, cada objeto, recebe um rótulo específico.

A rotulação das segmentações determina o número de componentes conexos existentes e a respectiva quantidade de pixels em cada componente. Na Figura 3.2, das doze FCNs que receberam a imagem, apenas quatro delas geraram segmentações positivas. A Figura 3.8 apresenta as segmentações positivas com rótulos para cada componente conexo obtido por meio da rotulação. Para simplificar a ilustração, os rótulos atribuídos aos componente da imagem estão identificados com “R1”, “R2”, “R3” e “R4”. Por exemplo, a segmentação referente ao arroz resultou em uma rotulação com dois componentes, enquanto o estrogonofe de frango resultou em apenas um componente. A segmentação do arroz é um Falso Positivo, uma vez que esse alimento não está presente na refeição do exemplo considerado. Identificar essa situação e descartar esse alimento caberá ao algoritmo.

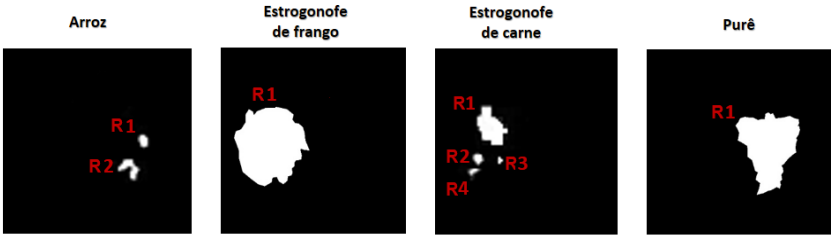


Figura 3.8 – Rotulação nas segmentações.

Etapa 2: Filtro dos rótulos

O filtro dos rótulos utiliza o resultado da etapa anterior, com informações sobre a quantidade de pixels em cada componente conexo, para manter em cada segmentação apenas o componente conexo que possui a maior quantidade de pixels. Esse processo mostrou-se eficiente na eliminação de possíveis segmentações secundárias que se comportam como ruído.

A Tabela 1 apresenta o número de pixels correspondentes a cada rótulo das segmentações da Figura 3.8. As segmentações resultantes do filtro são apresentadas na Figura 3.9. Essas segmentações correspondem aos maiores componentes conexos de cada segmentação.

Tabela 1 – Contagem dos pixels por rótulo.

Alimento	Número de pixels por rótulo				Maior
	R1	R2	R3	R4	
Arroz	42	91	–	–	91
Estrogonofe de frango	1853	–	–	–	1853
Estrogonofe de carne	427	34	5	23	427
Purê	1210	–	–	–	1210

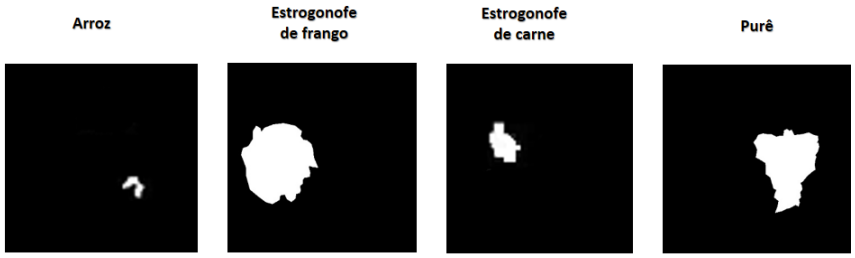


Figura 3.9 – Segmentações produzidas pela etapa filtro dos rótulos.

Etapa 3: Geração da matriz de interseção

A matriz de interseção M é uma matriz quadrada com ordem igual ao número de alimentos reconhecidos pelo modelo. Apesar dos nomes serem parecidos, a matriz de interseção não é a matriz de confusão utilizada pelas métricas de avaliação de um classificador. Neste contexto, essa matriz mostra quais FCNs geraram segmentações para o mesmo alimento. Isso é obtido por meio da operação de interseção entre as segmentações.

A Tabela 2 apresenta a matriz gerada para o exemplo dos alimentos na refeição da Figura 3.2. Neste exemplo, por questões de simplificação, estão sendo considerados apenas seis alimentos. Portanto, a matriz M possui seis colunas, uma para cada alimento do exemplo. O preenchimento dessa matriz é feito a partir da quantidade de pixels da interseção entre todas as segmentações. Essas interseções estão ilustradas na Figura 3.10.

Tabela 2 – Matriz de interseção M.

	Arroz	Estr. frango	Estr. carne	Purê	Maionese	Tomate
Arroz	91	0	0	91	0	0
Estr. frango	0	1853	427	0	0	0
Estr. carne	0	0	427	0	0	0
Purê	0	0	0	1210	0	0
Maionese	0	0	0	0	0	0
Tomate	0	0	0	0	0	0

A matriz de interseção contém informações sobre os alimentos que têm segmentações em comum. Na diagonal principal, é armazenado o total de pixels da segmentação do alimento correspondente a essa linha. As demais colunas armazenam o total de pixels da interseção entre o alimento da coluna e o alimento da linha. Por exemplo, a FCN do arroz gerou uma segmentação com 91 pixels. Na mesma linha, na coluna do purê, o valor 91 indica que essas segmentações se sobrepõem. Os valores iguais a zero indicam que os alimentos dessas colunas

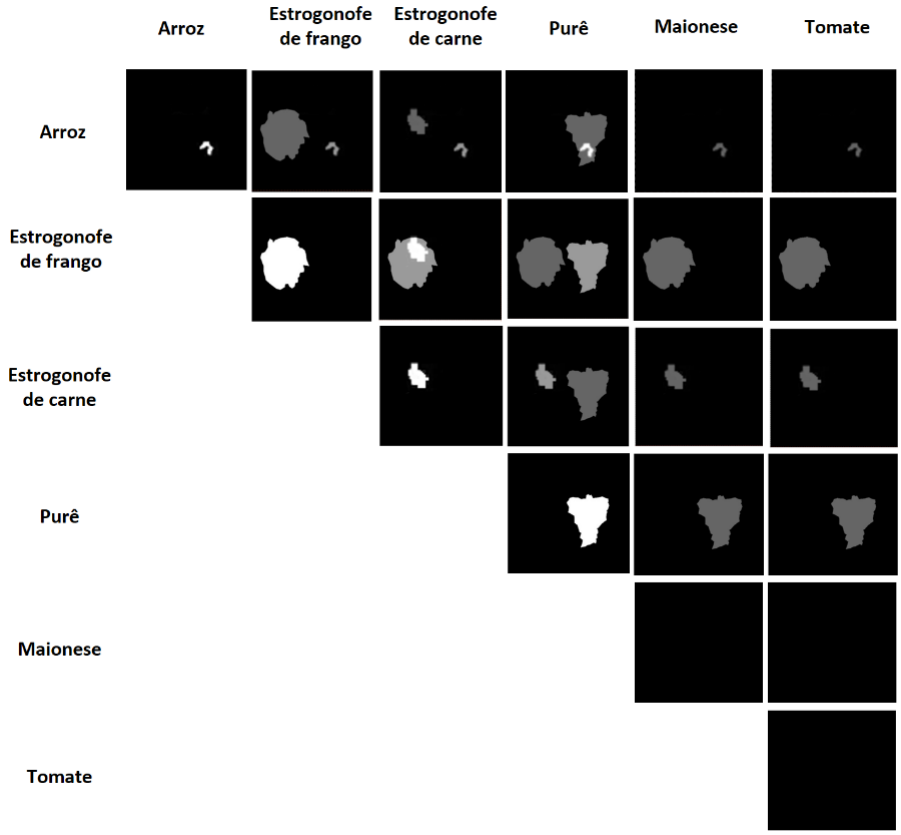


Figura 3.10 – Interseção entre as segmentações.

não têm interseção com o arroz.

Etapa 4: Processamento da matriz de interseção

A última etapa do algoritmo desenvolvido consiste no processamento da matriz de interseção M gerada nas etapas anteriores. Como a matriz M é simétrica, somente o triângulo superior é processado. Esse processamento determina quais alimentos estão presentes na refeição. A seguir, será apresentada a análise completa da matriz M da Tabela 2 pelo algoritmo.

O algoritmo inicia a interpretação na primeira linha, onde o alimento analisado é o arroz. A maior interseção do arroz com outro alimento nessa linha ocorre com o purê. Nesse caso, o arroz não será incluído como alimento reconhecido, uma vez que o purê apresenta uma segmentação maior (conforme dados da Tabela 1), ou seja, entre essas duas segmentações sobrepostas, a mais consistente é a do purê. O arroz é, então, descartado e não faz parte da lista de alimentos.

Na segunda linha, o alimento verificado é o estrogonofe de frango. Na mesma linha, a maior interseção ocorre com o estrogonofe de carne. Nesse caso, o estrogonofe de frango é incluído na lista de alimentos, uma vez que apresenta uma segmentação maior do que o estrogonofe de carne. O algoritmo ainda verifica se a segmentação do estrogonofe de carne tem um tamanho maior ou igual a 70% (valor definido de forma empírica) em relação ao tamanho da segmentação do estrogonofe de frango. Se isso ocorrer, o estrogonofe de carne é incluído como segunda opção na lista de alimentos. Analogamente, se existir uma segunda interseção, ela também é analisada

para verificar se será acrescentada como mais uma opção de alimento reconhecido. Quando um alimento é incluído como segunda ou terceira opção, ele não será mais analisado pelo algoritmo, e o valor correspondente na diagonal principal é modificado para 0 (zero).

O algoritmo continua, e o próximo alimento a ser analisado seria o estrogonofe de carne. No entanto, como esse alimento já foi analisado anteriormente, ele é descartado.

O próximo alimento analisado pelo algoritmo é o purê, que não tem interseção com nenhum outro alimento a ser considerada. O purê é então incluído na lista de alimentos. Quando o alimento em análise possui segmentação e, em sua linha, não existe nenhum outro alimento com valor de interseção, significa que esse alimento deve fazer parte da lista de alimentos. Nesse caso, a FCN desse alimento foi a única que obteve segmentação positiva e não apresenta semelhança com nenhum outro alimento na imagem da refeição apresentada.

Os dois últimos alimentos, a maionese e o tomate, não possuem segmentação, portanto, não serão comparados com a segmentação de nenhum outro alimento da sua linha e, consequentemente, não são incluídos na lista de alimentos.

Após todas as linhas serem processadas, o algoritmo é encerrado, retornando a lista de alimentos reconhecidos.

3.5 Considerações finais

Neste capítulo, foi apresentado o modelo de visão computacional desenvolvido, que se destaca pelo uso de um conjunto de FCNs para a segmentação dos alimentos e um algoritmo responsável por processar as segmentações e classificar os alimentos nas refeições, levando em consideração a semelhança entre eles. Esse algoritmo explora as características dos alimentos, resultando na criação de estruturas, como a matriz de interseção, que contêm informações determinísticas para a identificação dos alimentos realmente presentes na refeição.

O próximo capítulo abordará os testes e os resultados do modelo com base em dois conjuntos de treinamento diferentes dos utilizados para a implementação do modelo.

4 Resultados experimentais

Neste capítulo, são apresentados os resultados obtidos em dois experimentos. No primeiro, o modelo foi treinado utilizando um conjunto com 270 imagens de refeições contendo exemplos de alimentos divididos em 12 classes diferentes. Os resultados dos testes foram obtidos por meio da classificação de 108 imagens de refeições que não foram utilizadas no treinamento.

No segundo experimento, o conjunto de treinamento foi ampliado para 366 imagens de refeições, divididas em 20 classes, e o conjunto de teste contendo 236 imagens. Isso permitiu verificar a escalabilidade do modelo, ou seja, se a inclusão de novos alimentos na base de treinamento manteve a precisão na identificação dos alimentos.

O conjunto de treinamento foi criado a partir de imagens obtidas na web e rotuladas com a aplicação desenvolvida e apresentada na Seção 3.2. Os conjuntos de treinamento e teste foram armazenados em repositório de dados[1] estando disponível em [64]. A Figura 4.1 exibe alguns exemplos das imagens utilizadas. Todas as imagens das refeições estão sem fundo e, para cada refeição, está disponível uma imagem correspondente a máscara de segmentação para um alimento específico.

Figura 4.1 – Exemplos de imagens do conjunto de treinamento.

O aumento de dados foi aplicado a ambos os conjuntos. Como mencionado anteriormente, essa técnica foi utilizada para evitar o *overfitting*. Portanto, cada imagem foi replicada oito vezes e cada réplica foi rotacionada em 45 graus.

Para cada experimento, foram registrados os valores de acurácia e sensibilidade (Seção 2.7) obtidos. A acurácia é considerada uma das métricas mais simples e importantes, avaliando o percentual de acertos por meio da razão entre a quantidade de acertos e o total de entradas. A sensibilidade foi calculada, uma vez que essa métrica ganha importância em situações em que os Falsos Negativos são considerados mais prejudiciais do que os Falsos Positivos. No contexto deste trabalho, Falso Negativo significa não identificar um alimento presente na refeição, o que pode ser prejudicial na geração de informações sobre a refeição e seus alimentos.

[1] https://data.mendeley.com/datasets/7n36jtcpv3/2

Para a implementação do modelo foi utilizada a linguagem Python utilizando os recursos de hardware disponíveis em nuvem do *Google Colab* [54], juntamente com a biblioteca *Tensorflow* [65].

Todas as FCNs foram treinadas por 200 épocas utilizando a função de perda *entropia cruzada categórica* [66], juntamente com o otimizador *Adam* [44]. Números maiores de épocas não resultaram em melhoria na segmentação. O tempo de treinamento foi, em média, de trinta minutos para o primeiro experimento e sessenta minutos o segundo experimento.

4.1 Primeiro experimento

Neste experimento, o modelo foi treinado com 12 classes diferentes. A Tabela 3 apresenta as classes e a quantidade de exemplos de treinamento utilizados para cada uma delas.

Tabela 3 – Número de exemplos por alimento utilizado no treinamento do primeiro experimento.

Alimento	**Quantidade de Exemplos**
Arroz	80
Batata frita	53
Beterraba	35
Carne	138
Cenoura	64
Feijão	79
Macarrão	39
Maionese	13
Ovo	27
Purê de batata	31
Salada	95
Tomate	72

O treinamento das FCNs para segmentação desses alimentos foi realizado utilizando a arquitetura definida na Seção 3.3. A identificação dos alimentos é feita a partir do algoritmo apresentado na Seção 3.4. O Anexo A mostra as curvas de perda (*loss*) do histórico de treinamento das FCNs de todos os alimentos considerados nos dois experimentos. As redes FCN geram, então, as segmentações que são processadas pelo Algoritmo de Classificação de Alimentos.

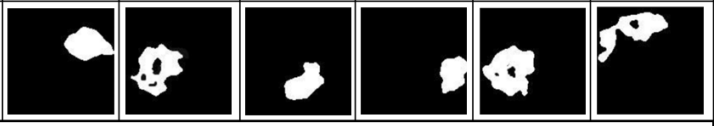
Após o treinamento, todo o conjunto de teste foi submetido ao modelo. As Figuras 4.2, 4.3 e 4.4 exibem as segmentações produzidas, juntamente com a lista de alimentos, para três imagens de refeições do conjunto de testes. Na Figura 4.2, observa-se que, para a refeição em questão, foram geradas seis segmentações positivas. O algoritmo de classificação, ao analisar essas segmentações, gerou como saída uma lista com cinco alimentos, identificados de ‘0’ a ‘4’. É possível notar que a FCN da batata frita gerou uma segmentação positiva para o macarrão. Essa segmentação ocorre devido à semelhança das imagens desses dois alimentos. Como a segmentação do macarrão é mais consistente do que a da batata frita, o algoritmo classificou esse alimento como macarrão, considerando a batata frita como a segunda opção. A refeitação da Figura 4.3 resultou em seis segmentações positivas. Novamente, as FCNs da batata frita e do

macarrão geraram segmentações para o mesmo alimento. A Figura 4.4 ilustra um exemplo de refeição com cinco alimentos classificados. Neste caso, não ocorreu interseção entre nenhuma das segmentações.

Refeição						
FCN	Batata Frita	Macarrão	Feijão	Carne	Salada	Tomate
Segmentações						

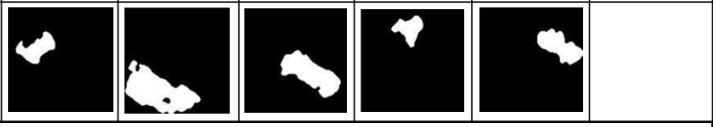
```
{'0': {'macarrão': 2943},'batatafrita': 2549},
 '1': {'feijão': 1503},
 '2': {'carne': 435},
 '3': {'salada': 878},
 '4': {'tomate': 402}}
```

Figura 4.2 – Primeiro exemplo de classificação de alimentos.

Refeição						
FCN	Arroz	Batata Frita	Carne	Feijão	Macarrão	Salada
Segmentações						

```
{'0': {'arroz': 1570},
 '1': {'carne': 1307},
 '2': {'feijão': 1099},
 '3': {'macarrão': 2482},'batatafrita': 2212},
 '4': {'salada': 1950}}
```

Figura 4.3 – Segundo exemplo de classificação de alimentos.

Refeição						
FCN	Arroz	Batata Frita	Carne	Salada	Tomate	
Segmentações						

```
{'0': {'arroz': 959},
 '1': {'batatafrita': 3022},
 '2': {'carne': 2111},
 '3': {'salada': 833},
 '4': {'tomate': 1375}}
```

Figura 4.4 – Terceiro exemplo de classificação de alimentos.

Para cada alimento classificado, é fornecido o tamanho da segmentação produzida. Por exemplo, a Figura 4.4, mostra que a segmentação produzida para a batata frita foi de 3022 pixels.

Essa informação pode ser utilizada para estimar a quantidade dos alimentos. Outra informação relevante é a segmentação produzida, que mostra a localização do alimento na refeição e pode ser usada como uma máscara combinada com a imagem da refeição para destacar o alimento classificado.

Para uma avaliação completa, todas as 180 imagens do conjunto de teste foram submetidas ao modelo treinado. A Tabela 4 apresenta o resultado completo das classificações. Para cada alimento testado, são fornecidos os valores das classificações, como Verdadeiro Positivo (VP), Verdadeiro Negativo (VN), Falso Positivo (FP) e Falso Negativo (FN). Com base nesses valores, foram calculadas duas métricas: acurácia e sensibilidade (Equações 2.6 e 2.7). O modelo alcançou uma acurácia média de 98% e uma sensibilidade média de 95%, demonstrando uma alta probabilidade de o modelo realizar classificações corretas dos alimentos.

Tabela 4 – Resultados do primeiro experimento.

Alimento	**Quantidade de testes por alimento**	**Classificação**				**Avaliação**	
		VP	**VN**	**FP**	**FN**	**Acurácia**	**Sensibilidade**
Arroz	38	38	67	3	0	0,97	1,00
Batata frita	18	16	88	2	2	0,96	0,89
Beterraba	21	19	87	0	2	0,98	0,90
Carne	42	39	66	0	3	0,97	0,93
Cenoura	15	14	93	0	1	0,99	0,93
Feijão	23	22	84	1	1	0,98	0,96
Macarrão	16	15	84	1	1	0,98	1,00
Maionese	14	13	94	0	1	0,99	0,93
Ovo	18	17	89	1	1	0,98	0,94
Purê de batata	22	22	83	3	0	0,97	1,00
Salada	26	25	82	0	1	0,99	0,96
Tomate	18	17	90	0	1	0,99	0,94
Média						**0,98**	**0,95**

4.2 Segundo experimento

O segundo experimento consiste no conjunto de treinamento expandido, elaborado a partir de 366 imagens de refeições que resultaram em 20 classes de alimentos. A Tabela 5 apresenta as classes e a quantidade de exemplos de treinamento utilizados para cada uma delas.

A ampliação das classes permitiu o reconhecimento mais específico de alguns alimentos. Por exemplo, no primeiro experimento, carnes de frango, boi e bife à milanesa eram classificadas em uma única classe, denominada carne. Neste segundo experimento expandido, esses três tipos de alimentos são reconhecidos separadamente. Outro exemplo é o macarrão e o espaguete, que no primeiro experimento eram classificados como macarrão, mas agora, neste segundo experimento, são classificados em classes diferentes.

Devido ao aumento do conjunto de dados, o modelo foi desafiado a classificar alimentos com forte similaridade. Forma, textura e cor podem variar tão pouco entre alguns alimentos a ponto de confundir até mesmo os seres humanos. Um exemplo disso é apresentado na Figura 4.5. Ao

Tabela 5 – Número de exemplos por alimento utilizado no treinamento do segundo experimento.

Alimento	Quantidade de Exemplos
Abóbora	31
Arroz	80
Batata frita	53
Beterraba	33
Bife à milanesa	22
Carne de boi	85
Carne de frango	55
Cenoura	70
Espaguete	30
Farofa	34
Feijão	80
Macarrão	26
Maionese	17
Mandioca	16
Milho	18
Ovo	27
Purê de batata	31
Repolho	18
Salada	95
Tomate	73

apresentar essa imagem ao modelo, a FCN da mandioca gera a segmentação correspondente, mas não é a única, pois a FCN da batata frita também gera uma segmentação para esse alimento. Essas segmentações são mostradas nas Figuras 4.6 (a) e (b). É possível observar a semelhança entre essas duas segmentações. A identificação do alimento é determinada pelo algoritmo de classificação, que analisa essas duas segmentações. Nesse caso, apesar da semelhança entre as segmentações, o algoritmo conclui que a segmentação da Figura 4.6(a) é a correta, pois possui uma segmentação ligeiramente mais consistente do que a da Figura 4.6(b). O algoritmo classifica, então, esse alimento como mandioca, com a batata frita sendo classificada como segunda opção.

Figura 4.5 – Mandioca.

O treinamento das FCNs para esse conjunto expandido de alimentos foi realizado nas mesmas condições do primeiro experimento e utilizando a mesma arquitetura. Após o treinamento das FCNs, o modelo foi submetido ao conjunto de testes com 236 imagens de refeições. A Tabela 6 apresenta os resultados do teste. Da mesma forma que no primeiro experimento, para todos os exemplos de alimentos são fornecidos os valores das classificações VP, VN, FP e FN. Com base nesses valores, foram calculadas as métricas de acurácia e sensibilidade.

Figura 4.6 – (a) Segmentação produzida pela FCN da mandioca. (b) Segmentação produzida pela FCN da batata frita.

Tabela 6 – Resultados do segundo experimento.

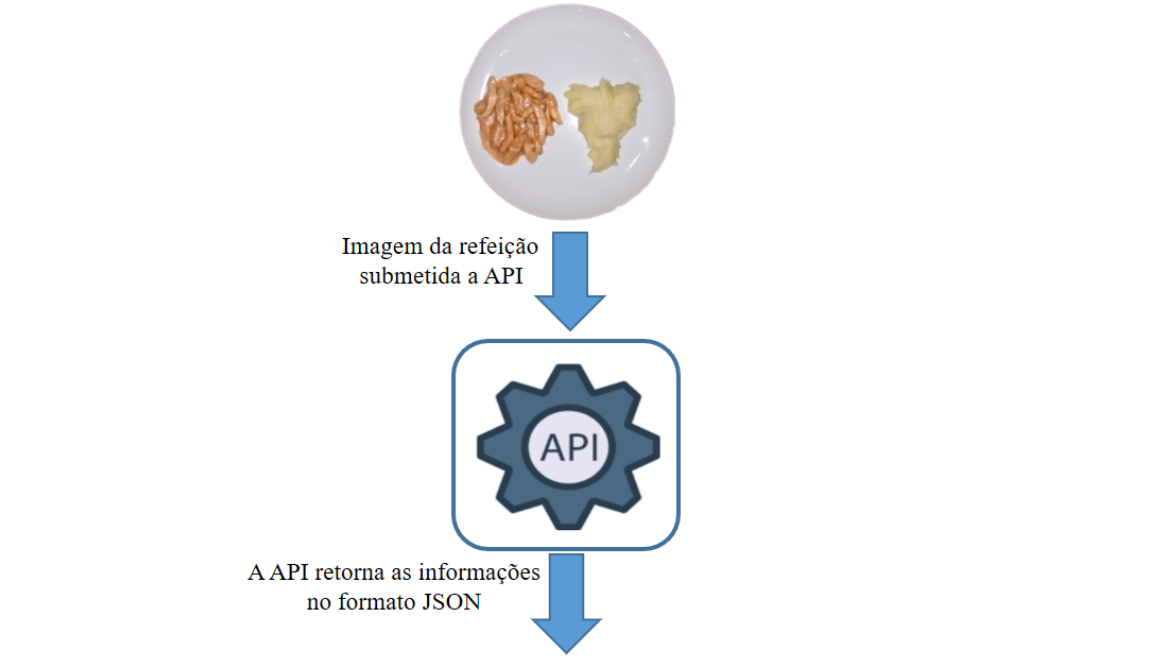
Alimento	**Quantidade de testes por alimento**	**Classificação**				**Avaliação**	
		VP	**VN**	**FP**	**FN**	**Acurácia**	**Sensibilidade**
Abóbora	25	24	206	5	1	0,975	0,960
Arroz	63	61	172	1	2	0,987	0,968
Batata frita	25	22	211	0	3	0,987	0,880
Beterraba	25	24	211	0	1	0,996	0,960
Bife à milanesa	18	17	217	1	1	0,992	0,944
Carne de boi	49	46	186	1	3	0,983	0,939
Carne de frango	34	31	199	3	3	0,975	0,912
Cenoura	21	17	214	1	4	0,979	0,810
Espaguete	18	18	215	3	0	0,987	1,000
Farofa	30	19	206	0	11	0,953	0,633
Feijão	49	37	186	1	12	0,945	0,755
Macarrão	16	14	218	2	2	0,983	0,875
Maionese	19	15	214	3	4	0,970	0,789
Mandioca	14	14	222	0	0	1,000	1,000
Milho	11	8	225	0	3	0,987	0,727
Ovo	24	19	212	0	5	0,979	0,792
Purê de batata	27	23	208	1	4	0,979	0,852
Repolho	17	15	218	1	2	0,987	0,882
Salada	54	54	182	0	0	1,000	1,000
Tomate	40	35	196	0	5	0,979	0,875
Média						**0,982**	**0,878**

Conforme a Tabela 6 apresenta, a acurácia média para os 20 alimentos é igual a 98,2%. Esse valor é muito próximo ao obtido no primeiro experimento, no qual esse valor foi de 98%. Já a sensibilidade teve um valor médio igual a 87,8%. Esse valor é menor do que o obtido no primeiro experimento, onde esse valor foi de 95%. A redução da sensibilidade ocorreu devido a alguns alimentos, como a farofa (63,3%) e o milho (72,7%). Para esses alimentos, o total de exemplos utilizados no treinamento foi relativamente pequeno, o que poderia justificar esses valores. No entanto, o mesmo não aconteceu com o repolho que possui um número de exemplos de treinamento ainda menor.

4.3 Experimentos com uma aplicação móvel

Em outro projeto de pesquisa, está sendo desenvolvida a aplicação móvel. Para isso, o modelo foi implementado e disponibilizado como uma API (*Application Programming Interface*), que contém as redes FCNs treinadas, juntamente com a implementação do algoritmo de classificação. A entrada dessa API é a imagem de uma refeição, e a saída é um arquivo JSON (*JavaScript Object Notation*) contendo a lista de alimentos classificados.

Na aplicação móvel, foram incluídas as funcionalidades básicas, como o cadastro de usuários e um histórico de refeições. No entanto, sua principal função é permitir que os usuários selecionem a imagem de uma refeição, submeta à API de reconhecimento e, depois disso, interprete o arquivo JSON retornado. A Figura 4.7 mostra um exemplo da refeição com os alimentos "Purê de batata" e "Estrogonofe de frango". No arquivo JSON, os campos de retorno contêm os nomes dos alimentos reconhecidos, as quantidades estimadas e as segmentações correspondentes. Todas as imagens estão codificadas no formato *base64*. Essa codificação não é mostrada na figura devido à extensão da sequência de caracteres, números e símbolos.

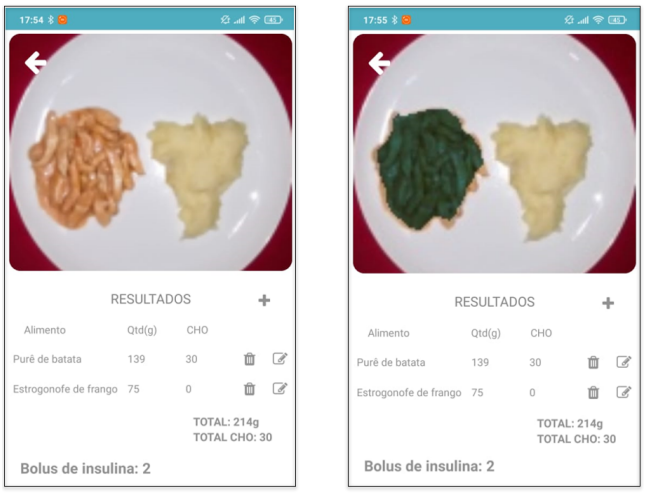



```
{"refeição": "imagem_refeição_base64",
 "dados_reconhecidos": [[{"nome_alimento": "Purê de batata",
                          "quantidade": "139",
                          "segmentação": "imagem_segmentação_pure_base64"}],
                        [{"nome_alimento": "Estrogonofe de frango",
                          "quantidade": "75",
                          "segmentação": "imagem_segmentação_estrogonofe_base64"}]
                        ]
}
```

Figura 4.7 – Funcionamento da API.

A partir das informações fornecidas pela API, a aplicação móvel realiza o cálculo total de carboidratos e determina a quantidade de insulina que deve ser administrada. A Figura 4.8

mostra a tela do aplicativo com a identificação dos alimentos na refeição apresentada como exemplo. Na refeição da Figura 4.8(b), o alimento “Estrogonofe de frango” é destacado na imagem quando o usuário seleciona o alimento na lista. Esse destaque é obtido a partir da segmentação do alimento correspondente, combinada com a imagem original. A Figura 4.9 apresenta mais dois exemplos de reconhecimento de alimentos.



(a) (b)

Figura 4.8 – (a) Tela do aplicativo com a lista de alimentos reconhecidos e suas informações. (b) Segmentação correspondente ao estrogonofe.

(a) (b)

Figura 4.9 – Exemplos de reconhecimento de alimentos pelo aplicativo.

4.4 Trabalhos relacionados

A comparação entre modelos de classificação que não utilizam as mesmas bases de dados pode não ser totalmente válida. No entanto, a Tabela 7 apresenta diversos modelos e suas respectivas acurácias, fornecendo um parâmetro para comparação.

Tabela 7 – Modelos para identificação de alimentos.

Modelo	**Conjunto de Dados**	**Acurácia**
Liu et al.[17]	UEC-256, 256 classes e 28375 imagens	87,0%
Reddy et al.[5]	20 classes e 10000 imagens	78,7%
Islam et al.[18]	Food-11, 11 classes e 23356 imagens	83,5%
Samraj et al.[19]	17 classes e 170 imagens	92,9%
Memis et al.[20]	UEC-100, 100 classes e 9060 imagens	87,5%
Chun et al.[15]	27 classes e 105427 imagens	82,0%
Freitas et al.[14]	9 classes e 1250 imagens	90,0%
Shiga [22]	16 classes e 1200 imagens	90,4%
Modelo proposto	**20 classes e 894 imagens**	**98,2%**

O desempenho do modelo proposto supera o de outros modelos, incluindo aqueles que não utilizaram bases de dados públicas, como Islam *et al.* [18], Chun *et al.* [15], Freitas *et al.* [14] e Shiga [22], sendo estes dois últimos voltados também para alimentos brasileiros. Embora Freitas *et al.* não tenha apresentado o valor exato da acurácia, o valor de 90% foi estimado com base nos resultados experimentais apresentados. Uma observação comum entre esses autores diz respeito à dificuldade de classificar refeições com vários alimentos, muitos dos quais têm imagens semelhantes.

O modelo proposto utiliza uma abordagem que conseguiu superar as dificuldades relatadas em todos os trabalhos relacionados. Vale destacar que, mesmo com o aumento da quantidade de alimentos para classificação, a acurácia manteve-se com valor elevado. Isto significa que a inserções de novos alimentos na base de treinamento manteve a precisão na identificação dos alimentos, para os testes realizados.

4.5 Considerações finais

Neste capítulo, os experimentos conduzidos permitiram a avaliação do desempenho do modelo proposto para a identificação de alimentos em refeições brasileiras. Apesar dos desafios inerentes à diversidade e à similaridade entre alguns alimentos, bem como a disponibilidade limitada de conjuntos de dados públicos abrangentes, os resultados obtidos validaram positivamente a eficácia do modelo. Ficou evidente que a precisão alcançada superou a de modelos anteriores, demonstrando a eficácia da abordagem adotada. Além disso, constatou-se que o modelo é escalável, permitindo a incorporação de novos alimentos sem prejudicar sua acurácia.

O próximo capítulo apresentará as conclusões finais deste estudo, resumindo os principais resultados e delineando as perspectivas para futuras pesquisas.

5 Conclusão

Neste estudo, foi abordado o desafio do reconhecimento automático de alimentos em refeições brasileiras. Para isso foi desenvolvido um modelo que tratou o problema da variação de aparência e a semelhança entre esses alimentos.

Ao longo deste estudo, foram definidos objetivos específicos, incluindo a criação de um conjunto de treinamento representativo com rótulos para cada alimento, o treinamento de um conjunto de FCNs para segmentar os alimentos nas refeições e a criação de um algoritmo para identificar os alimentos com base nas segmentações geradas.

Os resultados obtidos confirmam as hipóteses formuladas. Inicialmente, observou-se que a utilização de conjuntos de FCNs produziu segmentações que possibilitaram a identificação dos alimentos nas refeições pelo algoritmo de classificação desenvolvido. Esse algoritmo superou com sucesso o desafio da semelhança entre alimentos. Além disso, demonstrou-se que o modelo é escalável, permitindo a inclusão de novos alimentos na base de treinamento sem comprometer a eficiência do aprendizado ou a precisão na identificação.

Embora tenha sido alcançado sucesso nos objetivos e hipóteses deste estudo, reconhece-se que sempre há espaço para melhorias e desenvolvimentos futuros. Isso inclui a expansão do conjunto de alimentos reconhecidos, avaliação de diferentes técnicas de segmentação e aprimoramentos na eficiência computacional do modelo. No entanto, acredita-se que este trabalho representa um passo significativo em direção ao avanço da visão computacional aplicada à saúde e ao tratamento do diabetes.

Por fim, espera-se que este estudo inspire pesquisas adicionais nessa área fundamental da medicina, nutrição e tecnologia.

Publicação relacionada

Uma parte significativa dos resultados desta pesquisa foi publicada no artigo:

Carvalho, M.A., Pimenta, T.C., Silvério, A.C.P., Carvalho J.C.S. Computer vision model for food identification in meals from the segmentation obtained by a set of fully convolutional networks. *Journal of Ambient Intelligence and Humanized Computing* (2023). https://doi.org/10.1007/s12652-023-04703-9

Referências

[1] José E. P. de Oliveira et al. *Diretrizes da Sociedade Brasileira de Diabetes*. Clannad, São Paulo, 2017-2018.

[2] Sociedade Brasileira de Diabetes. *Manual de Contagem de Carboidratos para Pessoas com Diabetes*. Editora da Sociedade Brasileira de Diabetes, São Paulo, 2016.

[3] Mohammed A. Subhi and Sawal Md. Ali. A deep convolutional neural network for food detection and recognition. In *2018 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, pages 284–287, 2018.

[4] Meng-Lin Chiang et al. Food calorie and nutrition analysis system based on mask r-cnn. In *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*, pages 1721–1728, 2019.

[5] V. Hemalatha Reddy, Soumya Kumari, Vinitha Muralidharan, Karan Gigoo, and Bhushan S. Thakare. Food recognition and calorie measurement using image processing and convolutional neural network. In *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, pages 109–115, 2019. doi: 10.1109/RTEICT46194.2019.9016694.

[6] Murat Taskiran and Nihan Kahraman. Comparison of cnn tolerances to intra-class variety in food recognition. In *2019 IEEE International Symposium on Innovations in Intelligent Systems and Applications (INISTA)*, pages 1–5, 2019. doi: 10.1109/INISTA.2019.8778355.

[7] Parisa Pouladzadeh, Abdulsalam Yassine, and Shervin Shirmohammadi. Foodd: Food detection dataset for calorie measurement using food images. In Vittorio Murino, Enrico Puppo, Diego Sona, Marco Cristani, and Carlo Sansone, editors, *New Trends in Image Analysis and Processing - ICIAP 2015 Workshops*, pages 441–448, Cham, 2015. Springer International Publishing.

[8] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 - mining discriminative components with random forests. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision - ECCV 2014*, pages 446–461, Cham, 2014. Springer International Publishing.

[9] Ashutosh Singla, Lin Yuan, and Touradj Ebrahimi. Food/non-food image classification and food categorization using pre-trained googlenet model. In *Proceedings of the 2nd International Workshop on Multimedia Assisted Dietary Management*, pages 3–11, New York, NY, USA, 2016. Association for Computing Machinery. doi: 10.1145/2986035.2986039.

[10] Austin Myers, Nick Johnston, Vivek Rathod, Anoop Korattikara, Alex Gorban, Nathan Silberman, Sergio Guadarrama, George Papandreou, Jonathan Huang, and Kevin Murphy. Im2calories: Towards an automated mobile vision food diary. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1233–1241, 2015. doi: 10.1109/ICCV.2015.146.

[11] Parisa Pouladzadeh, Shervin Shirmohammadi, and Abdulsalam Yassine. You are what you eat: So measure what you eat! *IEEE Instrumentation & Measurement Magazine*, 19: 9–15, February 2016.

[12] Yoshiyuki Kawano and Keiji Yanai. Foodcam-256: A large-scale real-time mobile food recognitionsystem employing high-dimensional features and compression of classifier weights. In *Proceedings of the 22nd ACM International Conference on Multimedia*, pages 761–762, New York, NY, USA, 2014. Association for Computing Machinery. doi: 10.1145/2647868.2654869.

[13] Md Tohidul Islam, B.M. Nafiz Karim Siddique, Sagidur Rahman, and Taskeed Jabid. Food image classification with convolutional neural network. In *2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, volume 3, pages 257–262, 2018. doi: 10.1109/ICIIBMS.2018.8550005.

[14] Charles N. C. Freitas, Filipe R. Cordeiro, and Valmir Macario. Myfood: A food segmentation and classification system to aid nutritional monitoring. In *2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 234–239, 2020. doi: 10.1109/SIBGRAPI51738.2020.00039.

[15] Minki Chun, Hyeonhak Jeong, Hyunmin Lee, Taewon Yoo, and Hyunggu Jung. Development of korean food image classification model using public food image dataset and deep learning methods. *IEEE Access*, 10:128732–128741, 2022. doi: 10.1109/ACCESS.2022.3227796.

[16] Hokuto Kagaya, Kiyoharu Aizawa, and Makoto Ogawa. Food detection and recognition using convolutional neural network. In *Proceedings of the 22nd ACM International Conference on Multimedia*, pages 1085–1088, New York, NY, USA, 2014. Association for Computing Machinery. doi: 10.1145/2647868.2654970.

[17] Chang Liu, Yu Cao, Yan Luo, Guanling Chen, Vinod Vokkarane, Ma Yunsheng, Songqing Chen, and Peng Hou. A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure. *IEEE Transactions on Services Computing*, 11(2):249–261, 2018. doi: 10.1109/TSC.2017.2662008.

[18] Kh Tohidul Islam, Sudanthi Wijewickrema, Masud Pervez, and Stephen O'Leary. An exploration of deep transfer learning for food image classification. In *2018 Digital Image*

Computing: Techniques and Applications (DICTA), pages 1–5, 2018. doi: 10.1109/DICTA.2018.8615812.

[19] Andrews Samraj, Sowmiya D., Deepthisri K.A., and Oviya R. Food genre classification from food images by deep neural network with tensorflow and keras. In *2020 Seventh International Conference on Information Technology Trends (ITT)*, pages 228–231, 2020. doi: 10.1109/ITT51279.2020.9320870.

[20] Sefer Memis, Berker Arslan, Okan Zafer Batur, and Elena Battini Sönmez. A comparative study of deep learning methods on food classification problem. In *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, pages 1–4, 2020. doi: 10.1109/ASYU50717.2020.9259904.

[21] Takumi EGE and Keiji YANAI. Image-based food calorie estimation using recipe information. *IEICE Transactions on Information and Systems*, E101.D(5):1333–1341, 2018. doi: 10.1587/transinf.2017MVP0027.

[22] Y. M. Shiga. Sistema de identificação de alimentos baseado em imagens de porções alimentares. Dissertação de mestrado, Universidade Federal do Paraná, Paraná, 2015.

[23] Rafael C. Gonzales and Richard E. Woods. *Digital Image Processing*. Pearson Education, São Paulo, 3 edition, 2010.

[24] Hélio Pedrine and William Robson Schwartz. *Análise de Imagens Digitais: Princípios, Algoritmos e Aplicações*. Cengage Learning, São Paulo, 2008.

[25] Kenji Suzuki, Isao Horiba, and Noboru Sugie. Linear-time connected-component labeling based on sequential local operations. *Computer Vision and Image Understanding*, 89(1): 1–23, 2003. doi: 10.1016/S1077-3142(02)00030-9.

[26] Thomas H. Cormen et al. *Algoritmos: Teoria e Prática*. GEN LTC, São Paulo, 2012.

[27] Gianluigi Ciocca, Paolo Napoletano, and Raimondo Schettini. Food recognition: A new dataset, experiments, and results. *IEEE Journal of Biomedical and Health Informatics*, 21:588–598, May 2017.

[28] P. Norvig and S. Russell. *Inteligência Artificial*. Elsevier, 2013. ISBN 9788535237016.

[29] IBM. Introdução ao aprendizado de máquina, 2020. URL https://www.ibm.com/br-pt/analytics/machine-learning.

[30] Teressa B. Ludermir. Inteligência artificial e aprendizado de máquina: estado atual e tendências. *Estudos Avançados*, 35(101):85–94, abril 2021.

[31] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[32] A Géron. *Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & Tensorflow*. Alta Books, Rio de Janeiro, 2019.

[33] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous. *Bulletin of Mathematical Biophysics*, 52:99–115, 1943.

[34] Frank Rosenblatt. The perceptron – a perceiving and recognizing automaton. Technical Report Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

[35] Herbert Simon. *Redes Neurais*. Bookman, Porto Alegre, 2007.

[36] D. E. Rumelhart, J. L. McClelland, and P. R. Group. Learning representations by back-propagating errors. *Nature*, 323:533–6, 1986.

[37] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 1(521):436–444, 2015.

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[39] Data Science Academy. Deep learning book, 2021. URL https://www.deeplearningbook.com.br/.

[40] Dana Harry Ballard and Christopher M. Brown. *Computer Vision*. Prentice-Hall, 1982. ISBN 9780131653160, 0131653164.

[41] SERPRO. Visão computacional: O que é? como funciona?, 2021. URL https://www.serpro.gov.br/menu/noticias/noticias-2020/o-que-eh-visao-computacional.

[42] C. A. S. P. Rodrigues. Implementação de redes neurais convolucionais para a segmentação de imagens em tempo real com vistas à aplicação em robôs autônomos com dispositivos de visão de baixo custo. Dissertação de mestrado, Universidade Federal de Goiás (UFG), Goiânia, 2018.

[43] G. S. N. Hinton and S. K. Hinton. Neural networks for machine learning lecture 6a: Overview of mini-batch gradient descent, 2012. URL https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[44] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[45] Viceri-Seidor. Métricas de performance e funções de perda para machine learning, 2023. URL https://viceri.com.br/insights/metricas-de-performance-e-funcoes-de-perda-para-machine-learning/.

[46] A. M. J. F. P. s. A. Ribeiro. Um estudo comparativo entre cinco métodos de otimização aplicados em uma rnc voltada ao diagnóstico do glaucoma. *Revista de Sistemas e Computação*, 10(1), 2020.

[47] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015. doi: 10.1109/CVPR.2015.7298965.

[48] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[49] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 2481–2495, 1 Dec. 2017.

[50] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[51] M. Hollemans. Mobilenets do google no iphone, 2021. URL https://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/.

[52] TensorFlow Core. Image segmentation, 06 2022. URL https://www.tensorflow.org/tutorials/images/segmentation.

[53] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. doi: 10.1109/CVPR.2018.00474.

[54] Google. Colab, 2014. URL https://colab.research.google.com/notebooks/welcome.ipynb?hl=en.

[55] M Hollemans. Mobilenet version 2, 04 2018. URL https://machinethink.net/blog/mobilenet-v2/.

[56] D. Mariano. Métricas de avaliação em machine learning: acurácia, sensibilidade, precisão, especificidade e f-score. *BIOINFO - Revista Brasileira de Bioinformática e Biologia Computacional*, 2021.

[57] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "grabcut": Interactive foreground extraction using iterated graph cuts. In *ACM SIGGRAPH 2004 Papers*, pages 309–314, New York, NY, USA, 2004. Association for Computing Machinery. doi: 10.1145/1186562.1015720.

[58] Intel Corporation. Opencv:, 04 2000. URL https://docs.opencv.org.

[59] Nabil Ibtehaz and M. Sohel Rahman. Multiresunet : Rethinking the u-net architecture for multimodal biomedical image segmentation. *Neural Networks*, 121:74–87, 2020. doi: 10.1016/j.neunet.2019.08.025.

[60] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, 2017. doi: 10.1109/CVPR.2017.632.

[61] Xin Yu, Zhiding Yu, and Srikumar Ramalingam. Learning strict identity mappings in deep residual networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4432–4440, 2018. doi: 10.1109/CVPR.2018.00466.

[62] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016. URL http://arxiv.org/abs/1602.07261.

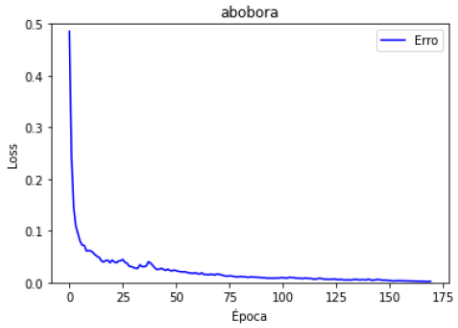
[63] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018. doi: 10.1109/CVPR.2018.00907.

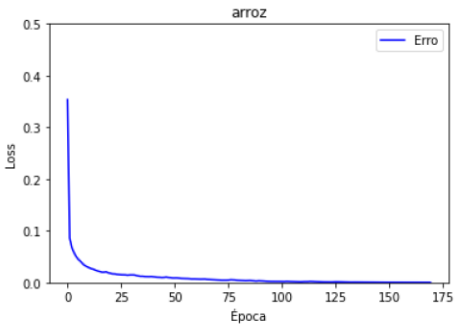
[64] Marcos Alberto Carvalho. Brazilian food images. Mendeley Data, 04 2023. URL https://data.mendeley.com/datasets/7n36jtcpv3/2.

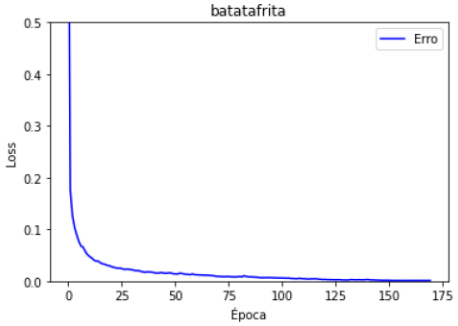
[65] Google Brain. Tensorflow, 11 2015. URL https://www.tensorflow.org/.

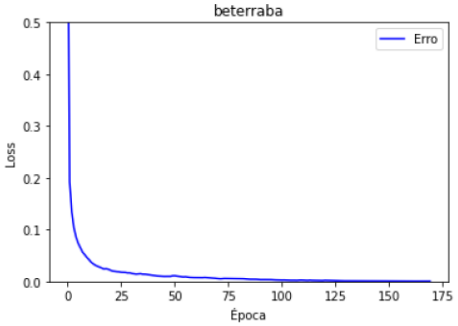
[66] Zhilu Zhang and Mert R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *CoRR*, abs/1805.07836, 2018.

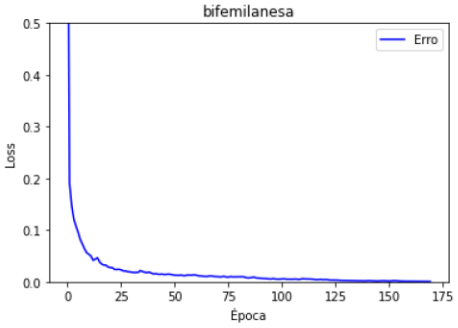
A Anexo: Curvas de perda (*loss*) do histórico de treinamento das FCNs

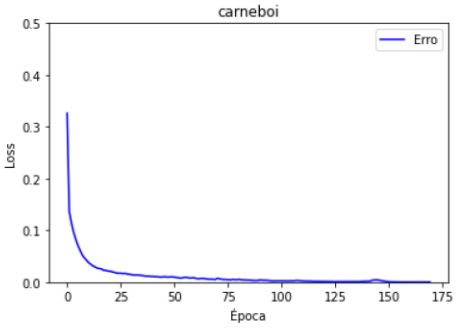


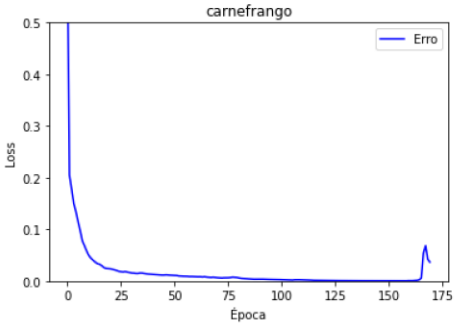


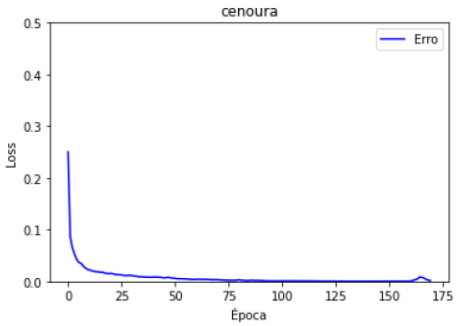


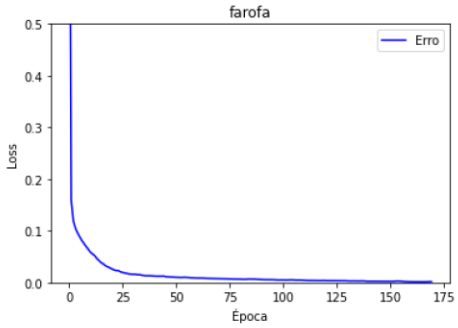


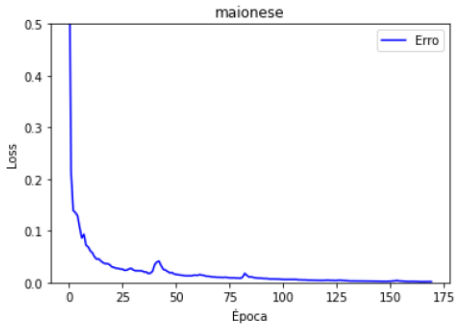


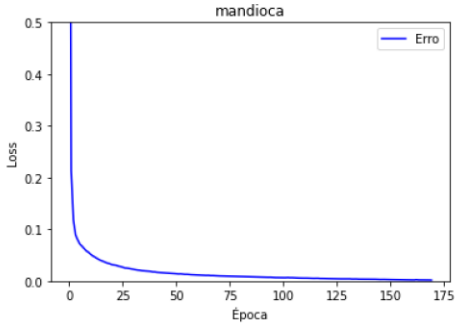


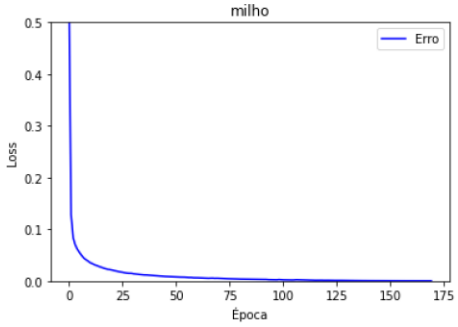


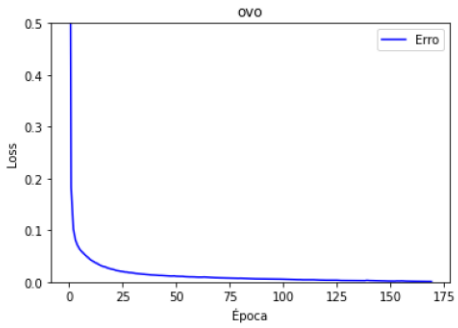


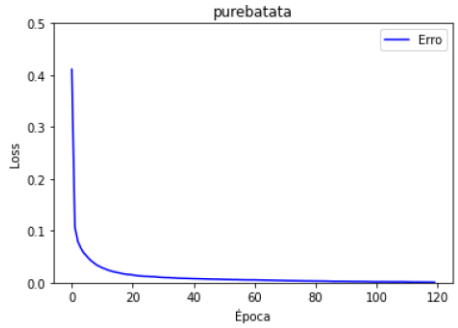

farofa
Erro
Loss
Época

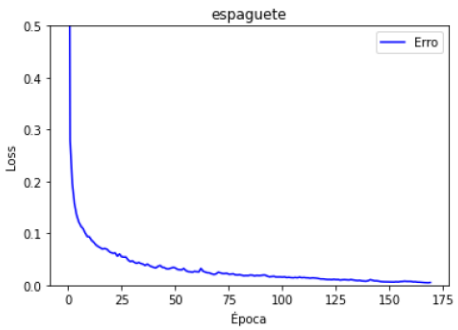

maionese
Erro
Loss
Época

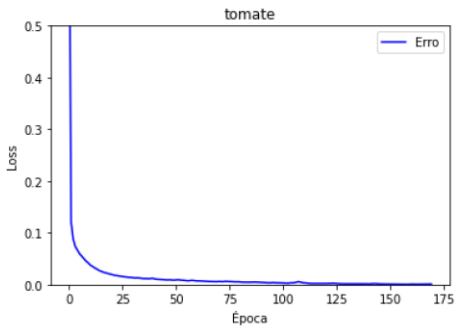

mandioca
Erro
Loss
Época

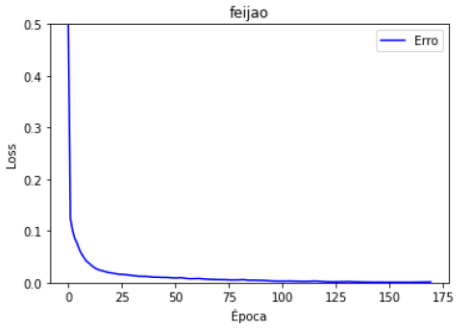

milho
Erro
Loss
Época

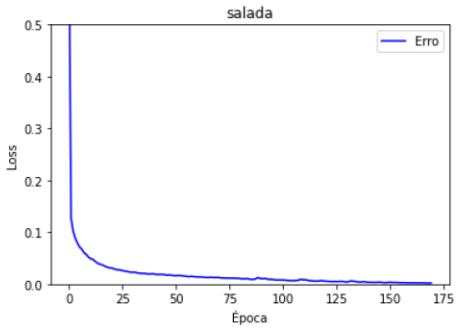

ovo
Erro
Loss
Época

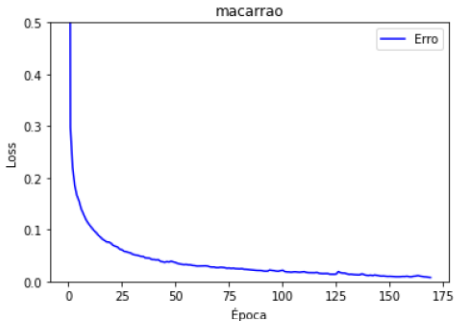

purebatata
Erro
Loss
Época

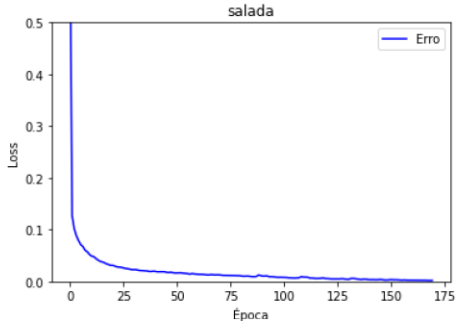

espaguete
Erro
Loss
Época


tomate
Erro
Loss
Época


feijao
Erro
Loss
Época
0.0
0.1
0.2
0.3
0.4
0.5
0
25
50
75
100
125
150
175


salada
Erro
Loss
Época
0.0
0.1
0.2
0.3
0.4
0.5
0
25
50
75
100
125
150
175


macarrao
Erro
Loss
Época
0.0
0.1
0.2
0.3
0.4
0.5
0
25
50
75
100
125
150
175


salada
Erro
Loss
Época
0.0
0.1
0.2
0.3
0.4
0.5
0
25
50
75
100
125
150
175