Trabalho de Conclusão de Curso

Avaliação de uma metodologia para recomendação de vestuário com base em similaridade utilizando a arquitetura YOLO e extração de características com DINOv2

Pedro Mateus Veras Simões
pmvs@ic.ufal.br

Orientador:

Thales Miranda de Almeida Vieira

Maceió, Novembro de 2024

Pedro Mateus Veras Simões

Avaliação de uma metodologia para recomendação de vestuário com base em similaridade utilizando a arquitetura YOLO e extração de características com DINOv2

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação do Instituto de Computação da Universidade Federal de Alagoas.

Orientador:

Thales Miranda de Almeida Vieira

Maceió, Novembro de 2024

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação do Instituto de Computação da Universidade Federal de Alagoas, aprovada pela comissão examinadora que abaixo assina.

Thales Miranda de Almeida Vieira - Orientador
Instituto de Computação
Universidade Federal de Alagoas

Evandro de Barros Costa - Examinador
Instituto de Computação
Universidade Federal de Alagoas

José Antão Beltrão Moura - Examinador
Departamento de Sistemas e Computação
Universidade Federal de Campina Grande

Maceió, Novembro de 2024

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecário: Valter dos Santos Andrade – CRB-1251

S593l Simões, Pedro Mateus Veras.
Avaliação de uma metodologia para recomendação de vestuário com base em similaridade utilizando a arquitetura YOLO e extração de características com DINOv2/ Pedro Mateus Veras Simões – 2024.
42 f : il.

Orientador: Thales Miranda de Almeida Vieira.
Monografia (Trabalho de Conclusão de Curso em Ciência da Computação) – Universidade Federal de Alagoas, Instituto de Computação, Maceió, 2024.

Bibliografia: f. 41-42.

1. Detecção de Objetos. 2. Yolo (Algoritmo). 3. Sistema de recomendação . Vestuário. 4. Similaridade. I. Título.

CDU: 004.352.242

Resumo

Um sistema de recomendação tem como objetivo sugerir novos itens ao usuário com base em suas preferências. Este trabalho propõe avaliar uma metodologia de recomendação de vestuário baseada em similaridade visual, na qual o usuário insere uma imagem de uma peça de roupa de interesse. O modelo de detecção de objetos YOLOv11 identifica todas as peças de vestuário presentes na imagem, e o usuário seleciona a peça de interesse, enquanto o modelo DINOv2 é utilizado para a extração de suas características visuais. Em seguida, o método envolve a realização de uma busca por peças similares na base de dados DeepFashion2, utilizando a distância euclidiana entre as características geradas. Os resultados mostram que o YOLOv11 obteve bons valores de mAP50, especialmente em experimentos com maior volume de dados de treino. No entanto, dificuldades foram encontradas na diferenciação de classes visualmente semelhantes, sugerindo possíveis ajustes na base de dados. O estudo conclui que a metodologia é viável para aplicação em sistemas de recomendação de vestuário baseados em imagens.

Palavras-chave: Detecção de Objetos; Sistema de recomendação de vestuário; Busca por similaridade.

Abstract

A recommendation system aims to suggest new items to the user based on their preferences. This work proposes evaluating a clothing recommendation methodology based on visual similarity, where the user inputs an image of a clothing item of interest. The YOLOv11 object detection model identifies all clothing items present in the image, and the user selects the item of interest, while the DINOv2 model is used for extracting its visual features. Next, the method involves performing a search for similar items in the DeepFashion2 database, using the Euclidean distance between the generated features. The results show that YOLOv11 achieved good mAP50 values, especially in experiments with a larger volume of training data. However, challenges were encountered in distinguishing visually similar classes, suggesting possible adjustments to the database. The study concludes that the methodology is viable for application in image-based clothing recommendation systems.

Keywords: Object Detection; Clothing Recommendation System; Similarity Search.

Conteúdo

Lista de Figuras

1

Introdução

1.1 Motivação

Sistemas de recomendação têm se tornado cada vez mais presentes no cotidiano, especialmente com o crescimento do comércio eletrônico. Esses sistemas auxiliam os usuários a encontrar produtos ou conteúdos de interesse em grandes bases de dados. Um dos métodos amplamente utilizados nesses sistemas é a busca por similaridade, que permite identificar itens semelhantes a um objeto de referência fornecido pelo usuário, facilitando o processo de recomendação de produtos.

Nesse contexto, o vestuário é um dos produtos mais comercializados online. Do ponto de vista da experiência do usuário que compra peças de roupa pela internet, é altamente desejável a possibilidade de encontrar uma peça de roupa similar a outra de referência, seja para receber recomendações de itens semelhantes ou para localizar roupas à venda que se assemelhem a uma peça que o usuário já possui. Essa necessidade está diretamente ligada ao desafio de encontrar as escolhas mais apropriadas diante da vasta gama de produtos disponíveis, algo que frequentemente sobrecarrega os usuários, como observado por (Ricci et al., 2010), que destacam que a abundância de escolhas pode, paradoxalmente, reduzir o bem-estar dos consumidores.

Para melhorar essa experiência, a busca visual pode ser uma ferramenta poderosa. (Jing et al., 2015) demonstraram que, com experimentos em produtos reais, os recursos de busca visual podem aumentar o engajamento dos usuários.

Os critérios usados para calcular a similaridade podem ser diversos, incluindo cor, textura, tipo de tecido e estilo. Essas informações, do tipo classe, necessitam ser anotadas para cada peça de roupa da base de dados. Há também atributos como imagem ou texto descritivo da peça de roupa que podem ser usados para calcular a similaridade. Em um sistema real, para se ter uma boa experiência do usuário, é interessante que o próprio usuário forneça uma imagem com um item de vestuário de interesse presente como mostrado na

figura 1.1. O sistema, em seguida, detecta todas as peças de roupa presentes na imagem, retornando ao usuário as opções detectadas nas quais ele escolherá a peça de interesse. O sistema, por sua vez, fará uma busca por itens visualmente semelhantes ao escolhido.

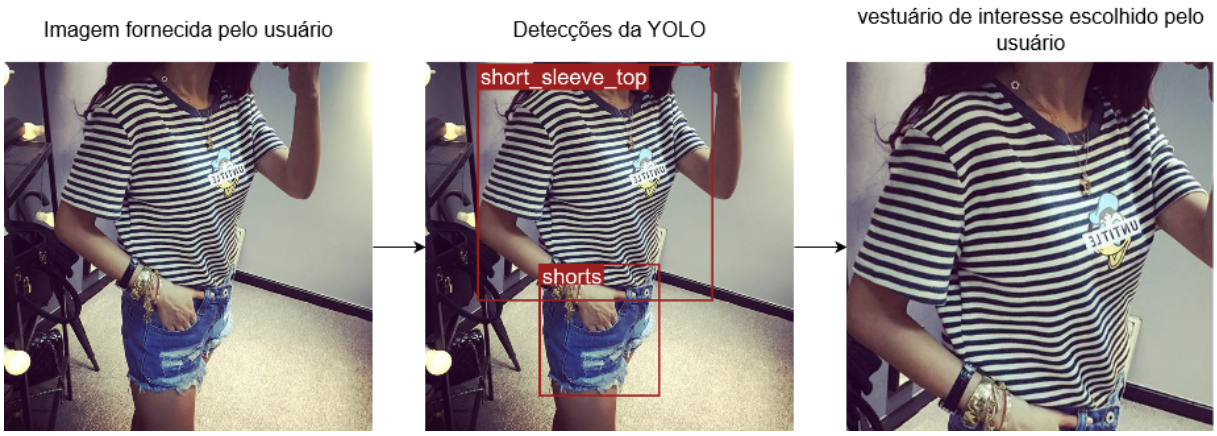

Figura 1.1: Diagrama da etapa de escolha do item de vestuário de interesse.

Na literatura, há diversos trabalhos sobre essa problemática, como o sistema de busca visual do Pinterest (Jing et al., 2015), que foi utilizado em produção, e um sistema de detecção e busca de itens de moda (Kucer and Murray, 2019), ambos utilizando um sistema dividido em etapas de detecção de objetos e então busca por similaridade do objeto detectado escolhido pelo usuário. No entanto, esses trabalhos foram publicados a alguns anos, e novos modelos mais sofisticados foram publicados desde então. Por isso, neste trabalho, iremos avaliar uma combinação de detector de objetos (Jocher and Qiu, 2024) e extrator de características mais recente (Oquab et al., 2023). Dessa forma, existe a necessidade do desenvolvimento de metodologias que façam uso de modelos de Visão Computacional mais recentes para alcançar resultados de maior qualidade para o usuário final em sistemas de recomendação de vestuário baseados em imagens.

1.2 Objetivo

O objetivo deste trabalho é desenvolver e avaliar uma metodologia que possa ser usada na construção de sistemas de recomendação de vestuário baseados em imagens, no qual o usuário pode fazer upload de uma imagem de interesse, o sistema detecta quais peças de roupas estão presentes na imagem, o usuário seleciona a peça de interesse e o sistema retorna a lista de vestuários mais visualmente semelhantes à peça selecionada. Além do desenvolvimento em si, o projeto também visa estudar os *trade-offs* entre diferentes possibilidades de tecnologias a serem utilizadas e analisar os resultados em conjunto com os casos de falha na etapa de detecção.

1.3 Solução proposta

Foi avaliada uma metodologia composta por três etapas: detecção, extração de características e busca. A detecção foi realizada por um modelo pré-treinado e ajustado (*fine-tuning*) utilizando o **Yolov11** (Jocher and Qiu, 2024), uma versão aprimorada do **Yolov4** (Bochkovskiy et al., 2020). Para o treinamento, foi utilizada a base de dados **DeepFashion2** (Ge et al., 2019). Na etapa de extração de características, o modelo pré-treinado **DinoV2** (Oquab et al., 2023) foi utilizado para extrair características da imagem de busca e para construir uma base de dados das características de todas as peças de roupas presentes na galeria. Para a busca, foi utilizada a técnica de busca por vizinhos mais próximos, onde as características extraídas das imagens são comparadas utilizando medidas de similaridade, nesse projeto a distância Euclidiana é utilizada, onde os n vizinhos mais próximos da imagem de busca são retornados. A análise de diferentes tamanhos de amostras do conjunto de treinamento foi realizada para estudar a quantidade mínima de dados de cada categoria que maximiza a performance do treinamento, visando minimizar o tamanho do conjunto de dados de treinamento e, consequentemente, os custos associados ao treinamento. A base de dados da galeria é composta por vetores de características extraídos do conjunto de dados **DeepFashion2**. Usando um modelo extrator de características chamado DinoV2 (Oquab et al., 2023), serão extraídos vetores de características (*embeddings*) de cada imagem da base. Esses embeddings são agrupados por categoria de vestuário, formando um arquivo para cada categoria. No processo de busca, o item de vestuário selecionado terá sua categoria correta associada, garantindo que apenas os embeddings dessa categoria sejam utilizados na busca, o que reduz o custo computacional. A busca pela peça mais similar é feita utilizando a distância euclidiana entre os embeddings.

1.4 Contribuições

Este trabalho contribui com a implementação e avaliação de uma metodologia para criar sistemas de recomendação de vestuário baseados em imagens usando modelos de visão computacional mais recentes como DinoV2 e YOLOv11.

2

Fundamentação Teórica

Neste capítulo, serão apresentados os fundamentos teóricos das técnicas utilizadas no projeto, abrangendo os seguintes temas: aprendizado profundo e redes neurais, redes neurais convolucionais, DinoV2, busca por similaridade com distância euclidiana, detecção de objetos, YOLO e métodos de análise e mensuração de resultados em detecção de objetos, como matriz de confusão, precisão, recall e mAP (*mean average precision*).

2.1 Aprendizado Profundo e Redes Neurais Artificiais

O aprendizado profundo é uma subárea do aprendizado de máquina que se destaca como um paradigma para reconhecer padrões complexos a partir de uma grande quantidade de dados, em uma variedade de problemas, incluindo visão computacional, como detecção de objetos, e processamento de linguagem natural, como tradução. Redes neurais profundas exploram a propriedade de que muitos sinais naturais são hierarquicamente compostos, onde características de nível superior são obtidas pela combinação de características de nível inferior. Em imagens, por exemplo, combinações locais de arestas formam padrões, padrões se organizam em partes e partes constituem objetos (LeCun et al., 2015).

O aprendizado profundo fornece uma estrutura muito poderosa para aprendizado supervisionado. Ao adicionar mais camadas e mais unidades dentro de uma camada, uma rede profunda pode representar funções de complexidade crescente. A maioria das tarefas que consistem em mapear um vetor de entrada para um vetor de saída, pode ser realizada por meio do aprendizado profundo, tendo-se modelos suficientemente grandes e conjuntos de dados de treinamento suficientemente grandes (Goodfellow et al., 2016).

Essa capacidade tem levado a recordes de desempenho em diversas áreas, como visão computacional e processamento de linguagem natural. Por exemplo, redes neurais convolucionais (CNNs) alcançaram resultados impressionantes em tarefas de classificação de

imagens no conjunto de dados ImageNet (Krizhevsky et al., 2012). No processamento de linguagem natural, modelos como o BERT (*Bidirectional Encoder Representations from Transformers*) (Devlin et al., 2018) redefiniram o estado da arte em diversas tarefas, como sumarização de documentos e tradução.

2.2 Fundamentos de redes neurais artificiais

As redes neurais artificiais são modelos computacionais inspirados no funcionamento dos neurônios biológicos. Cada neurônio recebe sinais de entrada, decide se deve ativá-los com base nesses sinais e, em seguida, transmite um sinal de saída para outros neurônios. Uma rede neural artificial é composta por uma rede de unidades interconectadas chamadas neurônios ou perceptrons, organizadas em camadas.

2.2.1 Perceptron

O perceptron, introduzido por (Rosenblatt, 1958), é um modelo de neurônio artificial que recebe entradas $x_1, x_2, ..., x_n$ ponderadas pelos pesos correspondentes $w_1, w_2, ..., w_n$. O neurônio soma esses produtos ponderados e adiciona um viés b para produzir uma saída que por sua vez é então passada por uma função de ativação ϕ para produzir a saída final do perceptron $\hat{y}$.

A saída de um perceptron é dada pela seguinte equação:

$$\hat{y} = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Onde:

- $\hat{y}$ é a saída do perceptron,
- x_i são as entradas,
- w_i são os pesos correspondentes,
- b é o viés,
- ϕ é a função de ativação.

A função de ativação geralmente é uma função não linear, como:

- A função sigmoide: $\sigma(x) = \frac{1}{1+e^{-x}}$
- A função ReLU (*Rectified Linear Unit*): $\text{ReLU}(x) = \max(0, x)$

A função de ativação é usada para introduzir não linearidade no modelo, permitindo que o perceptron resolva problemas que não são linearmente separáveis.

2.2.2 Multilayer Perceptron (Perceptron Multicamadas)

O Multilayer Perceptron (MLP), ou Perceptron Multicamadas, é uma arquitetura de redes neurais bastante utilizada, que consiste na presença de múltiplas camadas de neurônios totalmente conectados, incluindo uma camada de entrada, outra de saída e uma ou mais camadas ocultas. O processo de propagação pela frente (*feedforward*) na MLP envolve a passagem dos sinais de uma camada a outra, no qual o sinal de saída de uma camada serve de entrada para a próxima camada e assim sucessivamente.

Cada neurônio em uma camada recebe uma combinação linear das saídas de todos os neurônios da camada anterior, aplica uma função de ativação para introduzir não linearidade e, em seguida, passa o resultado para todos os neurônios da camada subsequente. Este processo continua até que o sinal alcance a camada de saída. A Figura 2.1 ilustra um diagrama de um Multilayer Perceptron (MLP) com 3 neurônios na camada de entrada, 2 camadas ocultas com 5 neurônios cada e, por fim, 2 neurônios na camada de saída.

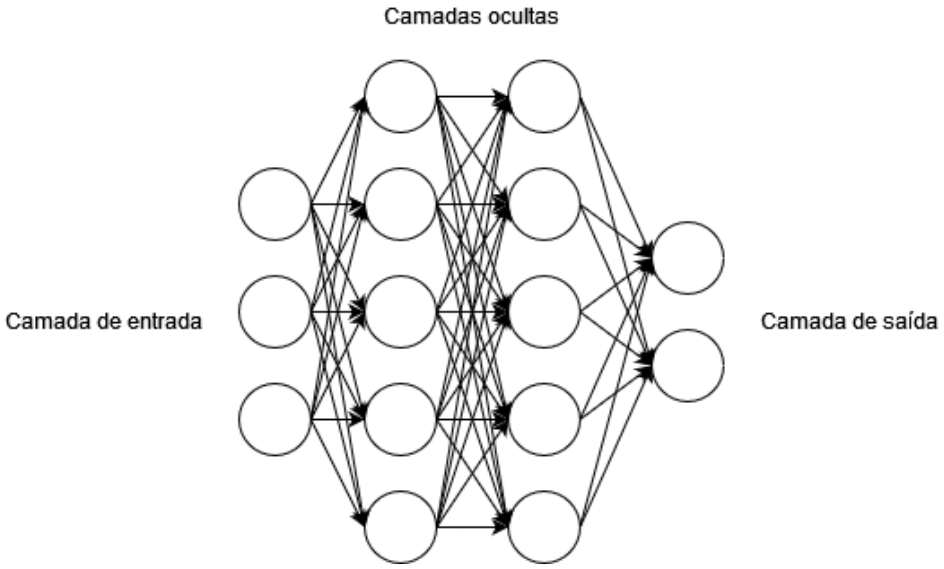

Figura 2.1: Diagrama de um Multilayer Perceptron (MLP).

2.2.3 Treinamento de Redes Neurais

O treinamento de uma rede neural envolve a atualização iterativa dos pesos W para reduzir a diferença entre a saída prevista $\hat{y}$ pela rede e a saída desejada y. Essa diferença é calculada por uma função de custo $J(\theta)$, que mede o erro da rede neural.

A função de custo $J(\theta)$ pode ser representada da seguinte forma:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(y_i, \hat{y}_i)$$

onde:

- θ são os parâmetros da rede neural ou pesos,
- m é o número de exemplos de treinamento,
- $L(y_i, \hat{y}_i)$ é a função de perda que mede a diferença entre a previsão $\hat{y}_i$ e a verdadeira saída y_i.

Durante o treinamento, uma quantidade de exemplos da base de dados de treinamento é selecionada, os pesos da rede são iniciados, normalmente usando números aleatórios, os dados de entrada são fornecidos para a camada de entrada da rede e a saída da rede é calculada a partir da propagação para frente (*forward propagation*), no qual os sinais de saída da camada $a^{(l-1)}$ são passados como entrada para a próxima camada l, e assim sucessivamente até se obter os sinais de saída da rede $\hat{y}$, que será utilizado para calcular a função de custo $J(\theta)$.

O algoritmo de retropropagação (*backpropagation*), introduzido por (Rumelhart et al., 1986), é uma técnica de treinamento de redes neurais que ajusta os pesos da rede para minimizar a função de custo. Ele calcula os gradientes da função de custo em relação a cada peso da rede usando a regra da cadeia e, então, atualiza os pesos na direção do gradiente negativo usando uma técnica de otimização chamada de gradiente descendente.

Para atualizar os pesos, o algoritmo de gradiente descendente é usado. A atualização é feita na direção do gradiente negativo da função de custo:

$$W \leftarrow W - \eta \frac{\partial J}{\partial W}$$

onde:

- η é a taxa de aprendizado,
- $\frac{\partial J}{\partial W}$ é o gradiente da função de custo em relação aos pesos.

Esses gradientes são usados para atualizar os pesos da rede neural, permitindo que ela aprenda a partir dos dados de treinamento ao encontrar iterativamento o conjunto ideal de pesos W que minimiza uma função de custo J.

Enquanto o método do gradiente descendente padrão e a retropropagação formam a base do treinamento de redes neurais, abordagens mais modernas como o algoritmo Adam (*Adaptive Moment Estimation*) (Kingma and Ba, 2014) têm sido amplamente adotadas para melhorar a eficiência e a convergência do treinamento.

2.3 Redes neurais convolucionais

As redes neurais convolucionais (CNNs) são um tipo específico de redes neurais projetadas para processar imagens. Elas têm sido bastante eficientes e apresentaram desempenho superior em uma variedade de problemas de visão computacional, como classificação de imagens, segmentação semântica e detecção de objetos.

De acordo com (LeCun et al., 1998), existem duas principais motivações para não usar redes totalmente conectadas tradicionais para imagens. Primeiro, imagens normalmente possuem muitas dimensões. Por exemplo, uma imagem de tamanho 768x768 teria um total de 589824 dimensões apenas de entrada para a rede neural, o que levaria uma rede neural totalmente conectada a conter muitos pesos e, consequentemente, um maior custo computacional. Segundo, redes totalmente conectadas não possuem invariância a translações e distorções locais das entradas. Isso significa que pequenas variações na posição dos objetos na imagem poderiam levar a grandes variações nas respostas da rede. As CNNs resolvem esse problema aplicando filtros que detectam características locais, tornando a rede mais robusta a mudanças na posição dos objetos.

2.3.1 Arquitetura das Redes Convolucionais

A arquitetura de uma CNN é composta por várias camadas diferentes, cada uma com um papel específico:

1. **Camada Convolucional**: A camada convolucional é a principal camada das CNNs. Ela aplica um conjunto de filtros (*kernels*) à imagem de entrada para extrair características locais, como bordas, texturas e padrões. A operação de convolução para imagens discretas pode ser descrita matematicamente como:

$$(K * I)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i+m-\lfloor M/2 \rfloor, j+n-\lfloor N/2 \rfloor) K(m,n)$$

Onde:

- I é a imagem de entrada,
- K é o kernel (filtro), com dimensões $M \times N$,
- (i,j) são as coordenadas do pixel na imagem de saída,
- m,n são índices inteiros que percorrem as dimensões do kernel.

Por exemplo, em uma imagem de entrada de 32×32, ao aplicar um kernel de 5×5, a saída será de tamanho 28×28, pois a aplicação do filtro reduz cada dimensão em 4 pixels (2 de cada lado), considerando uma operação sem *padding*.

2. **Camada de Pooling**: A camada de pooling, que é utilizada após as camadas convolucionais, reduz as dimensões espaciais da entrada, agregando valores locais. Isso ajuda a diminuir a quantidade de parâmetros e o custo computacional, além de ajudar "a tornar a representação aproximadamente invariante a pequenas translações da entrada" (Goodfellow et al., 2016). A operação de pooling mais comum é o max pooling, onde o pixel (i,j) do resultado é obtido a partir do valor máximo de uma janela que desliza pela imagem com um determinado passo. Por exemplo, ao aplicar uma operação de max pooling com tamanho 2×2 e passo 2 em uma imagem de 28×28, a saída será de 14×14, pois cada janela de 2×2 pixels é reduzida a um único valor (o máximo).

3. **Camada Totalmente Conectada**: Nas etapas finais de uma CNN, são usadas camadas totalmente conectadas, onde cada neurônio está conectado a todos os neurônios da camada anterior. Essas camadas são usadas para combinar as características extraídas e realizar a classificação final. Por exemplo, suponha que o vetor achatado de uma camada anterior tenha dimensão 1×784 (14x14x4), ele é conectado a um número de neurônios correspondente ao número de classes de saída no caso de um problema de classificação.

Um exemplo notável de uma rede CNN é a LeNet-5, uma das primeiras CNNs de sucesso, projetada para reconhecimento de dígitos escritos à mão. Ela consiste em camadas convolucionais, seguidas por camadas de pooling, e termina com camadas totalmente conectadas. Essa arquitetura foi revolucionária e demonstrou a eficácia das CNNs em tarefas de reconhecimento de padrões em imagens (LeCun et al., 1998). Na Figura 2.2 está a arquitetura da LeNet-5 original.

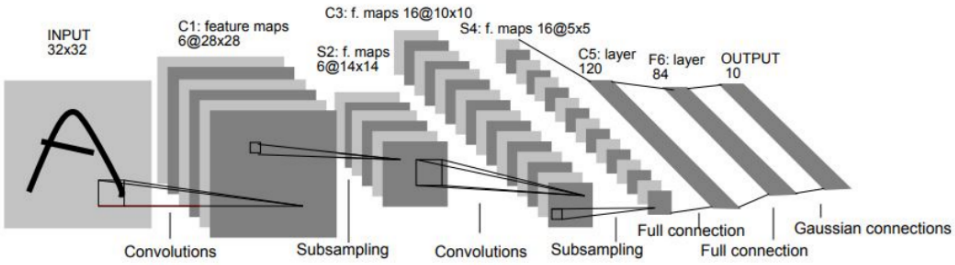

Figura 2.2: Arquitetura da LeNet-5 uma Rede Neural Convolucional (LeCun et al., 1998).

2.4 DinoV2: Um Modelo de fundação para Extração de Características

O DinoV2 (Oquab et al., 2023) é um modelo de fundação recente em visão computacional, tendo uma capacidade de extrair representações generalizáveis de imagens, servindo como modelo base para diversas tarefas de visão computacional. Diferente das redes convolucionais tradicionais, o DinoV2 utiliza aprendizado auto-supervisionado, permitindo seu treinamento em grandes volumes de dados sem a necessidade de rótulos. Essa abordagem o torna útil para problemas de visão computacional que necessitam de um extrator de características.

No contexto de busca por similaridade utilizando a distância euclidiana, o DinoV2 gera vetores de características que representam as imagens em um espaço latente. Essas representações compactas capturam as características visuais dos objetos, permitindo comparações eficazes. Assim, a similaridade entre itens pode ser medida diretamente pela distância euclidiana entre os *embeddings*, o que facilita a identificação de itens visualmente similares.

2.4.1 Busca por Similaridade com Distância Euclidiana

A distância euclidiana é uma das métricas mais utilizadas para medir a distância entre dois vetores em um espaço vetorial. Quando aplicada em visão computacional, a distância eu-

clidiana é usada para comparar os vetores de características gerados por modelos de *deep learning* como o DinoV2 a partir de imagens. Esses *embeddings* são representações numéricas compactas e representativas das informações visuais presentes em uma imagem, como padrões, texturas e formas.

Após a extração dos *embeddings* de duas ou mais imagens, a distância euclidiana pode ser utilizada para medir o grau de similaridade entre elas. A fórmula para calcular a distância euclidiana entre dois vetores $\mathbf{x}$ e $\mathbf{y}$, ambos de dimensão n, é dada por:

$$d(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Onde x_i e y_i são as coordenadas dos vetores $\mathbf{x}$ e $\mathbf{y}$, que representam as imagens no espaço de características gerado pelo DinoV2. Quanto menor for o valor de $d(\mathbf{x},\mathbf{y})$, mais semelhantes são as imagens.

2.5 Detecção de objetos e YOLO

A detecção de objetos é uma tarefa de visão computacional que envolve a localização e identificação de objetos em uma imagem. Essa tarefa é mais complexa do que a simples classificação de imagens, pois requer não apenas a identificação da classe do objeto, mas também a determinação da posição exata do objeto na imagem. Para realizar essa tarefa de maneira eficiente, várias abordagens foram desenvolvidas ao longo dos anos, entre as quais a *You Only Look Once* (YOLO) se destaca por sua performance e precisão.

2.5.1 Métodos de Detecção de Objetos

Antes do YOLO, uma das abordagens mais populares para detecção de objetos era a Region-based Convolutional Neural Networks (R-CNN). A R-CNN propõe regiões de interesse e aplica uma rede neural convolucional para extrair características dessas regiões. As principais variantes da R-CNN incluem:

- **R-CNN**: Utiliza uma técnica seletiva de busca para gerar cerca de 2000 regiões de proposta de objetos e aplica uma CNN para extrair características de cada região (Girshick et al., 2015).
- **Fast R-CNN**: Melhora a R-CNN ao realizar a extração de características em uma única passagem pela imagem, em vez de extrair características de cada região proposta separadamente (Girshick, 2015).
- **Faster R-CNN**: Introduz a Rede de Proposta de Região (RPN), que gera propostas de objetos de maneira eficiente e integrada (Ren et al., 2015).

Embora esses métodos tenham melhorado a precisão da detecção de objetos, eles ainda são computacionalmente intensivos e lentos para aplicações em tempo real.

2.5.2 Arquitetura da YOLO

YOLO é uma das arquiteturas mais populares para detecção de objetos introduzido por (Redmon et al., 2016), a abordagem YOLO reformula a tarefa de detecção de objetos como um problema de regressão de fim a fim, ao invés de usar métodos baseados em regiões. A principal vantagem do YOLO é sua capacidade de realizar detecção de objetos em tempo real com alta precisão. A Figura 2.3 apresenta a arquitetura da YOLOv8.

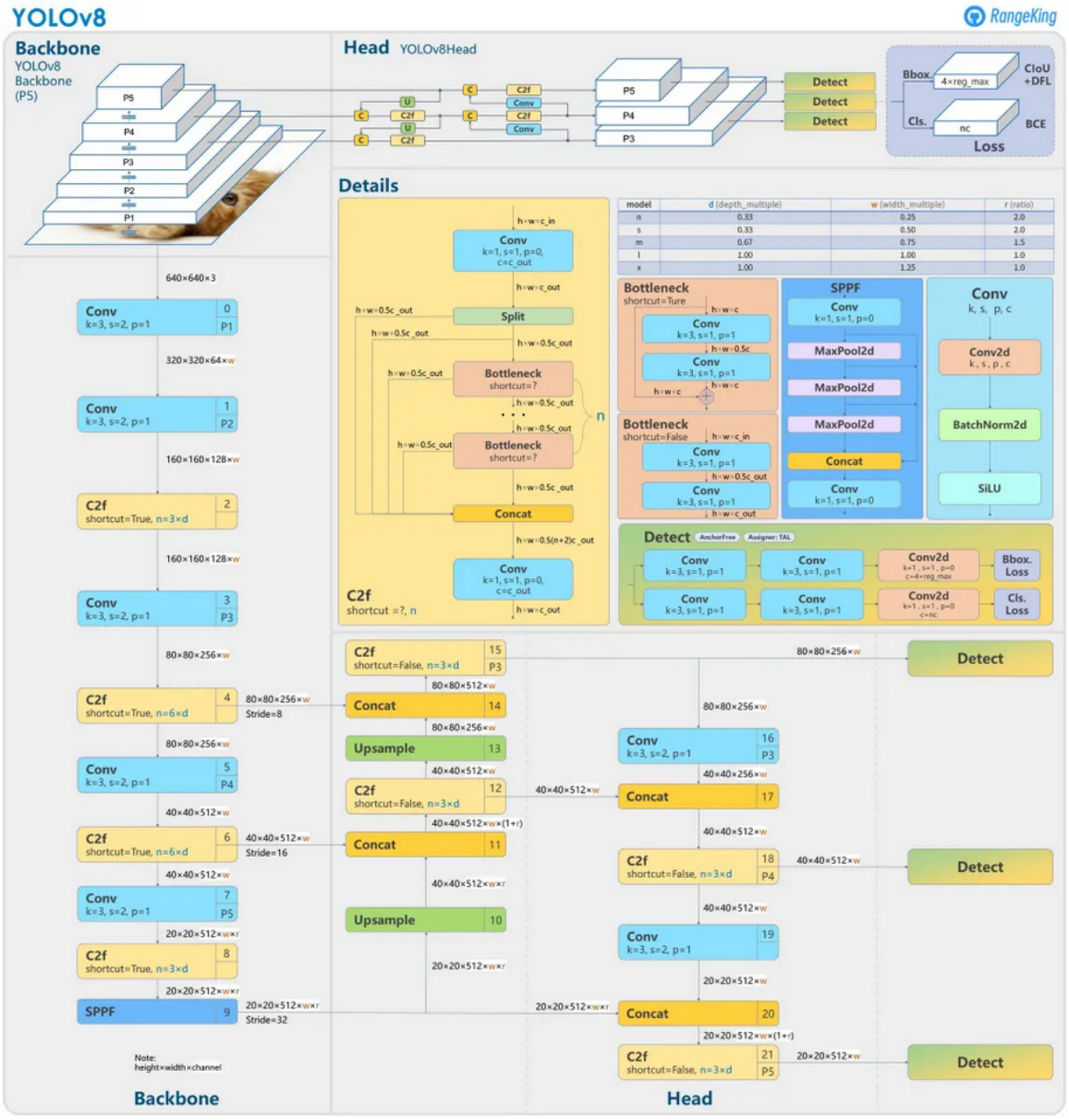

Figura 2.3: Arquitetura da YOLOv8. (Solawetz and Francesco)

A arquitetura YOLO divide a imagem de entrada em uma grade $S \times S$. Para cada célula da grade, o modelo prevê B caixas delimitadoras (*bounding boxes*) e C probabilidades de classes. Cada caixa delimitadora é representada por cinco valores: x, y, w, h e o score de confiança. Onde, (x, y) são as coordenadas do centro da caixa, w e h são a largura e a altura, e a confiança representa a precisão da caixa contendo um objeto. A saída da rede para uma imagem de entrada pode ser representada como um tensor tridimensional de dimensões $S \times S \times (B \times 5 + C)$. A Figura 2.4 apresenta a sequência de funcionamento da YOLO com a divisão da imagem em grade, a previsão das caixas delimitadoras e a identificação dos objetos.

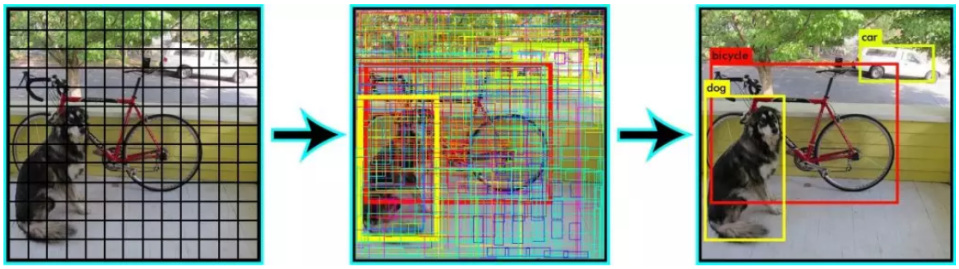

Figura 2.4: Exemplo de detecção de objetos usando YOLO.

2.5.3 YOLOv11

O YOLOv11 (Jocher and Qiu, 2024) é uma nova versão que traz melhorias em relação às versões anteriores, tornando-o mais eficiente e preciso para tarefas de detecção de objetos. A seguir, destacam-se as principais características do modelo:

Extração de Características Aprimorada: O YOLOv11 utiliza uma arquitetura aprimorada de *backbone* e *neck*, que melhora as capacidades de extração de características, resultando em uma detecção de objetos mais precisa e um desempenho superior em tarefas complexas.

Otimização para Eficiência e Velocidade: O modelo introduz um design arquitetônico refinado e pipelines de treinamento otimizados, oferecendo velocidades de processamento mais rápidas, enquanto mantém um equilíbrio ideal entre precisão e desempenho.

Maior Precisão com Menos Parâmetros: Com os avanços no design do modelo, o YOLOv11m alcança uma maior precisão média (mAP) no conjunto de dados COCO, utilizando 22% menos parâmetros em comparação ao YOLOv8m. Isso torna o modelo computacionalmente eficiente sem comprometer sua precisão.

2.6 Métodos de Avaliação

A avaliação dos modelos de detecção de objetos é fundamentalmente importante para entender a performance a fim de comparar o impacto de diferentes arquiteturas e diferentes bases de dados nos resultados obtidos, além de ajudar na análise dos casos onde o modelo mais erra. Algumas das técnicas de avaliação mais usadas no problema de detecção de objetos são precisão, recall, *mean average precision* (mAP) e matriz de confusão.

As métricas precisão e recall são complementares para analisar os resultados de um modelo de detecção de objetos. A precisão mede a proporção de verdadeiros positivos em relação ao total de predições positivas feitas pelo modelo (ou seja, a soma de verdadeiros positivos e falsos positivos). No contexto de detecção de objetos, ela indica o quão preciso é o modelo ao prever a presença de um objeto de uma determinada classe. Uma alta precisão significa que a maioria dos objetos detectados pelo modelo são realmente verdadeiros obje-

tos daquela classe. O recall mede a proporção de verdadeiros positivos em relação ao total de instâncias reais de positivos (ou seja, a soma de verdadeiros positivos e falsos negativos). No contexto de detecção de objetos, ela indica a capacidade do modelo de encontrar todos os objetos de uma determinada classe. Um alto recall significa que a maioria dos objetos daquela classe presentes na imagem foram corretamente identificados pelo modelo. Suas fórmulas são:

$$\text{Precisão} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos + Falsos Positivos}}$$

$$\text{Recall} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos + Falsos Negativos}}$$

O *Mean Average Precision* (mAP) é uma métrica utilizada para avaliar a precisão do modelo de detecção de objetos. Ele é calculado pela média das precisões médias de cada classe, dado pela seguinte fórmula:

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i$$

Onde N é o número total de classes, e AP_i é a precisão média para a classe i.

A matriz de confusão é uma tabela amplamente utilizada para avaliar o desempenho de modelos de classificação. Para problemas de detecção de objetos, ela é usada para mostrar o número de verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos para cada classe de objeto.

Definições:

- **Verdadeiros Positivos**: O modelo prevê corretamente a presença de um objeto de uma determinada classe.
- **Verdadeiros Negativos**: O modelo prevê corretamente a ausência de um objeto de uma determinada classe.
- **Falsos Positivos**: O modelo prevê incorretamente a presença de um objeto de uma determinada classe.
- **Falsos Negativos**: O modelo não consegue prever a presença de um objeto de uma determinada classe que realmente está presente.

A matriz de confusão oferece uma visão detalhada sobre o desempenho do modelo de detecção de objetos, permitindo a identificação de padrões de erro específicos, como a tendência do modelo de confundir uma classe com outra. Essas informações são cruciais para a melhoria contínua do modelo.

3

Metodologia

A metodologia utilizada no projeto consiste na análise da base de dados DeepFashion2, o processo de treinamento do modelo de detecção de objetos, a análise dos resultados obtidos e o sistema completo com as etapas de detecção de objetos e busca de imagens semelhantes.

As etapas de detecção de objetos e busca de imagens foram escolhidas separadamente devido à presença de mais de uma peça de roupa na imagem de interesse. Isso é necessário para proporcionar uma boa experiência ao usuário, permitindo que ele escolha qual peça de roupa será utilizada na busca. Para a etapa de detecção de objetos, o modelo YOLOv11, sendo a versão mais recente do YOLO, foi selecionado devido à sua alta performance e baixo custo computacional, sendo amplamente utilizado nesse tipo de tarefa. Já para a etapa de extração de características, o DINOv2 foi escolhido por apresentar excelentes resultados em diversas tarefas de visão computacional que utilizam características extraídas de imagens.

3.1 Base de Dados

A base de dados utilizada neste projeto foi a DeepFashion2 (Redmon et al., 2016) devido à sua ampla variedade de imagens e categorias de roupas, sendo uma base de dados bastante utilizada para tarefas de moda. Além disso, contém diversas anotações como *bounding boxes*, segmentação e outras.

A base de dados consiste em três diretórios: treino, teste e validação. Devido à parte de treino ser suficiente para este projeto, este foi o segmento selecionado. Portanto, sempre que for citada a base de dados do projeto, estará se referindo ao segmento de treino da DeepFashion2 apenas.

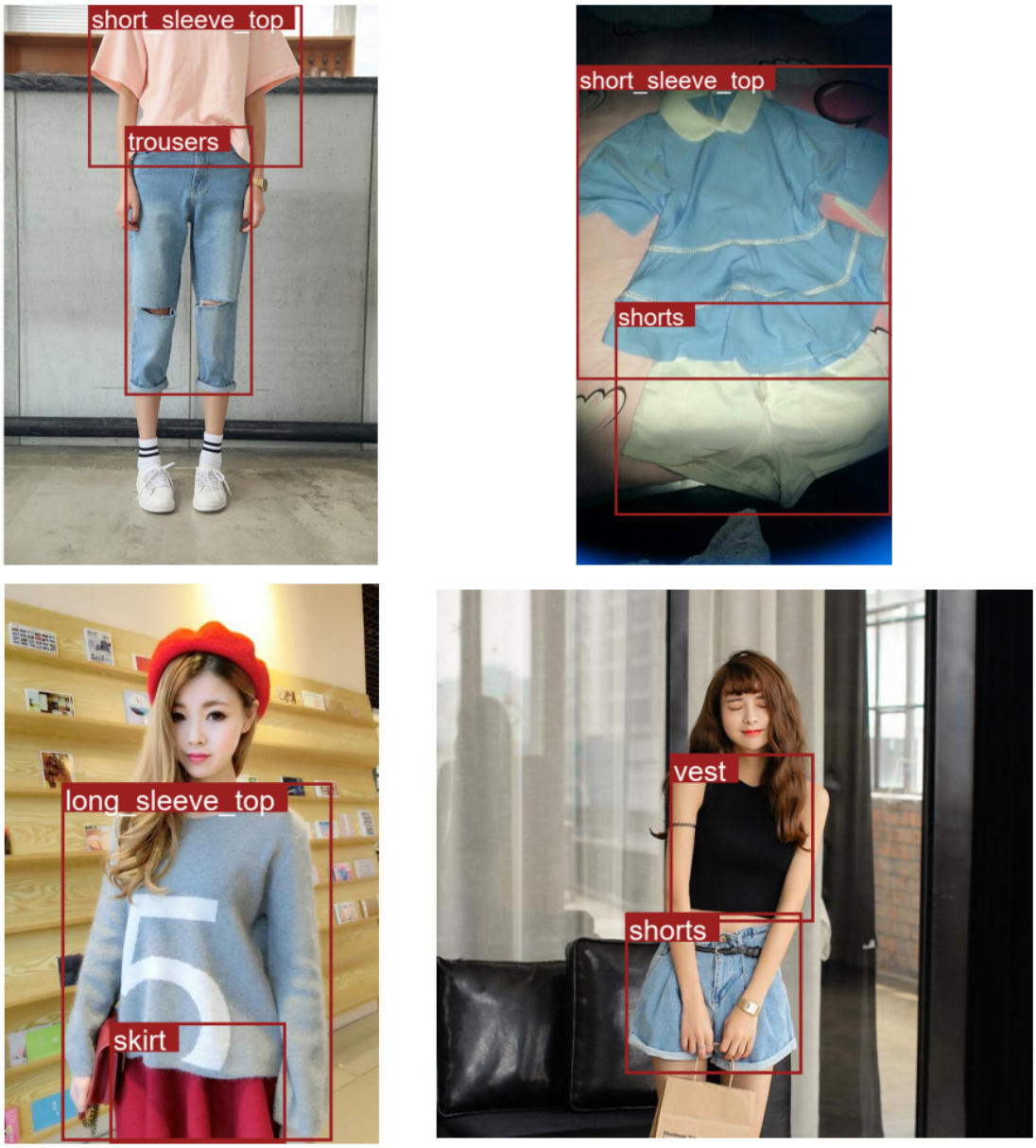

Figura 3.1: Exemplos de imagens da base de dados DeepFashion2 com detecção de *bounding boxes* das roupas.

3.1.1 Distribuição da Base de Dados:

Os dados da base utilizados para o projeto são imagens e anotações das *bounding boxes* presentes na imagem de acordo com 13 diferentes categorias de roupas, sendo elas e suas respectivas quantidades:

- Short Sleeve Top (Blusas de Manga Curta): 71.645 objetos
- Trousers (Calças): 55.387 objetos

- Shorts (Shorts): 36.616 objetos
- Long Sleeve Top (Blusas de Manga Longa): 36.064 objetos
- Skirt (Saias): 30.835 objetos
- Vest Dress (Vestidos em Estilo de Colete): 17.949 objetos
- Short Sleeve Dress (Vestidos de Manga Curta): 17.211 objetos
- Vest (Coletes): 16.095 objetos
- Long Sleeve Outwear (Casacos de Manga Longa): 13.457 objetos
- Long Sleeve Dress (Vestidos de Manga Longa): 7.907 objetos
- Sling Dress (Vestidos de Alça): 6.492 objetos
- Sling (Blusas de Alça): 1.985 objetos
- Short Sleeve Outwear (Casacos de Manga Curta): 543 objetos

Totalizando 312.186 objetos distribuídos entre um total de 191.961 imagens. A seguir, na figura 3.2, o histograma da distribuição das categorias:

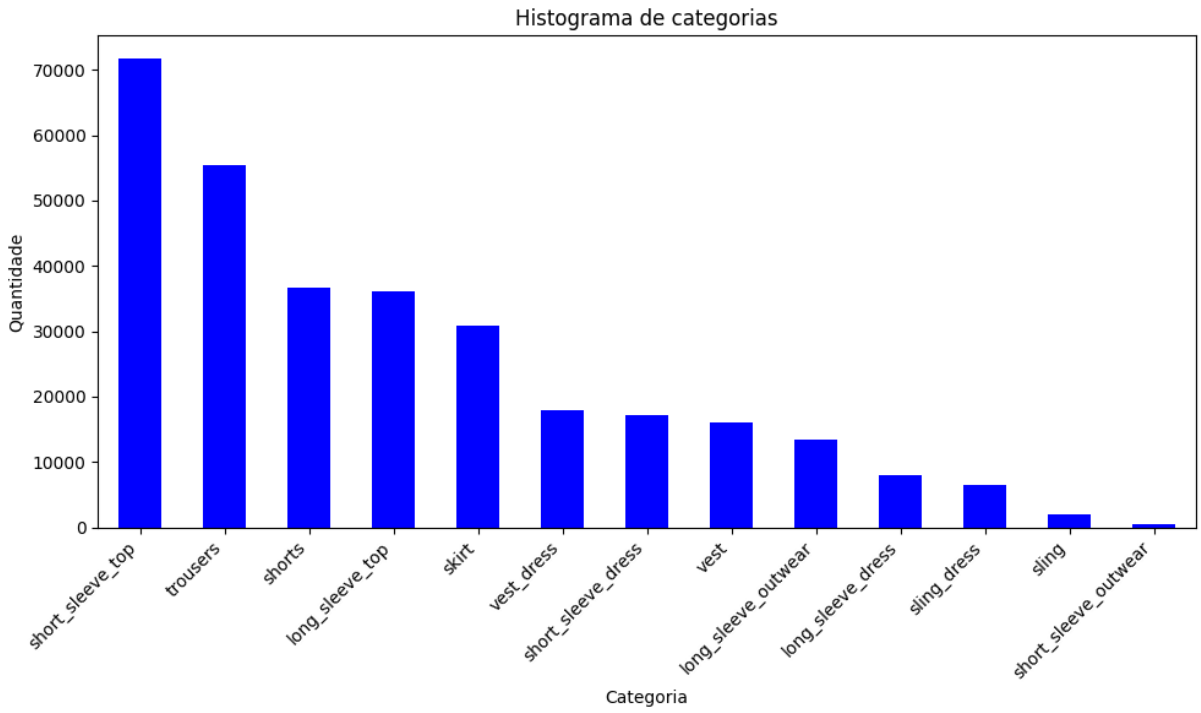

Figura 3.2: Histograma da distribuição das categorias.

A seguir, na figura 3.3, um exemplo de uma imagem da base de dados com as *bounding boxes* selecionadas:

Figura 3.3: Exemplo de imagem com *bounding boxes* selecionadas.

Para avaliar se a quantidade de objetos em cada categoria influencia a qualidade das peças de roupa recuperadas pelo sistema, foram criadas três versões de uma nova base de dados a partir de uma amostragem aleatória da base de treino da DeepFashion2. Na primeira versão, foram escolhidos de forma aleatória 250 objetos de cada categoria no máximo. Como cada objeto está em uma imagem, porém cada imagem pode conter múltiplos objetos, os outros objetos presentes nas imagens escolhidas também são adicionados na nova base de dados. Para a segunda e terceira versões, foram escolhidos 500 e 1000 objetos por categoria, respectivamente. Para a finalidade do projeto, as novas versões da base de dados foram divididas entre 70% para treino, 15% para teste e 15% para validação. Por fim, foram removidas imagens presentes em mais de uma das seções de treino, teste e validação, para

evitar distorções na fase de validação do modelo de detecção de objetos.

Tabela 3.1: Comparação da quantidade de objetos de cada categoria presentes nos experimentos

Categoria	**Experimento 1**	**Experimento 2**	**Experimento 3**
Short Sleeve Top	756	1545	3029
Trousers	733	1496	2946
Shorts	657	1250	2343
Skirt	536	1154	2169
Long Sleeve Top	461	909	1772
Vest	359	696	1404
Long Sleeve Outwear	322	650	1309
Vest Dress	303	608	1158
Sling Dress	282	557	1109
Short Sleeve Dress	259	538	1073
Long Sleeve Dress	262	530	1055
Sling	268	521	1031
Short Sleeve Outwear	255	501	543

3.2 Treinamento do Detector de Objetos

Três modelos YOLOv11 foram treinados, cada um com uma versão diferente da base de dados, por um total de 50 épocas. O experimento 1, usando a versão 1 da base, utilizou 3242 imagens e 5453 objetos. O experimento 2, com a versão 2, usou 6464 imagens e 10955 objetos. Já o experimento 3, com a versão 3, utilizou 12411 imagens e 20941 objetos.

3.3 Avaliação do Modelo de Detecção de Objetos

Para avaliar a performance do modelo YOLOv11 de detecção de objetos, serão analisados e comparados a progressão da métrica mAP50 no segmento de validação da base de cada experimento ao longo das épocas a fim de verificar visualmente qual experimento converge mais rápido. Também serão analisadas a métrica mAP50 de cada categoria e de cada experimento em conjunto com a matriz de confusão, e por fim, serão analisados alguns casos onde o modelo mais erra usando os dados presentes na matriz de confusão para identificar as principais razões dos erros.

3.4 Diagrama Completo do Sistema

A metodologia consiste em três etapas: a etapa de detecção de objetos, na qual o usuário seleciona uma imagem de interesse contendo peças de vestuário; a etapa de extração de

características; e a etapa de busca por similaridade.

3.4.1 Detecção de Objetos com YOLOv11

Na primeira etapa, o sistema utiliza o modelo YOLOv11 para detectar as peças de vestuário presentes na imagem de interesse. O modelo retorna as *bounding boxes* e as respectivas categorias dos objetos detectados, podendo ser nenhum, um ou mais de um. O usuário seleciona apenas uma peça de interesse das que foram detectadas para a busca por similaridade. Apenas a peça selecionada será usada nas etapas subsequentes, juntamente com sua categoria e a respectiva *bounding box* no formato $(x_{min}, y_{min}, x_{max}, y_{max})$, onde:

- x_{min} e y_{min} representam as coordenadas do canto superior esquerdo da caixa delimitadora;
- x_{max} e y_{max} representam as coordenadas do canto inferior direito da caixa delimitadora.

3.4.2 Extração de Características com DINOv2

Na segunda etapa, uma vez selecionada a peça de roupa, o sistema realiza um recorte da região correspondente à *bounding box* e extrai um vetor de características da imagem utilizando o modelo DINOv2. Este modelo gera um vetor no espaço $\mathbb{R}^{768}$, que representa a imagem de forma compacta em termos de suas características visuais.

3.4.3 Busca por Similaridade usando Distância Euclidiana

Na terceira etapa, o sistema realiza a busca por similaridade. Cada categoria de vestuário possui um arquivo pré-processado contendo uma árvore KD-Tree com os *embeddings* de todas as peças daquela categoria presentes na base de dados. O sistema carrega a árvore correspondente à categoria da peça de interesse. A busca por similaridade é feita calculando a distância euclidiana entre o *embedding* da peça de roupa recortada e os *embeddings* armazenados na KD-Tree. Esse processo permite identificar as n peças de roupa mais similares à peça de interesse de mesma categoria, com as menores distâncias indicando maior similaridade visual.

O diagrama completo do sistema pode ser visualizado a seguir na figura 3.4:

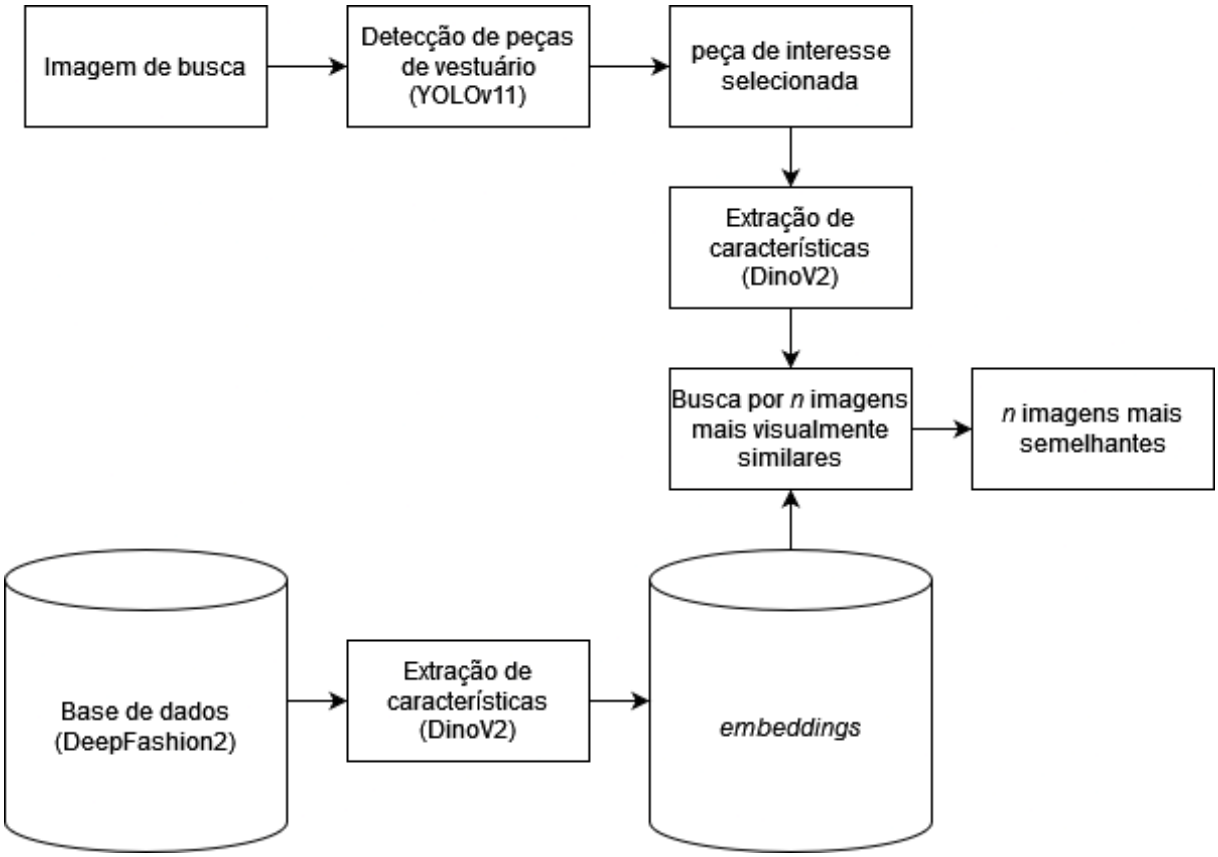

Figura 3.4: Diagrama completo do sistema.

4

Experimentos

4.1 Treinamento do Modelo

O treinamento do modelo YOLO foi realizado com os hiperparâmetros padrão da próprio YOLO, utilizando a técnica de *fine-tuning* com o modelo base YOLOv11l, que possui um total de 25.320.503 parâmetros. O treinamento foi conduzido com um tamanho de lote (batch size) de 4, executado em uma GPU RTX 2060 com 6 GB de memória VRAM, utilizando a biblioteca PyTorch e com *mixed precision* ativado. O experimento 1 teve a duração total de 2,023 horas de treinamento, enquanto os experimentos 2 e 3 demandaram, respectivamente, 3,939 e 7,617 horas de treinamento. Após cada época de treinamento, o modelo foi avaliado calculando-se o mAP50 na base de validação, permitindo a análise do desempenho ao longo das épocas. O mAP50 é o mAP calculada com um limiar de IoU de 0.5.

A Figura 4.1 apresenta o gráfico que compara o progresso do treinamento nos três experimentos, onde o eixo horizontal representa as épocas e o eixo vertical o mAP50. Como é possível verificar na Figura 4.1, o experimento com maior número de imagens na base de treino converge para um mAP50 maior em uma menor quantidade de épocas.

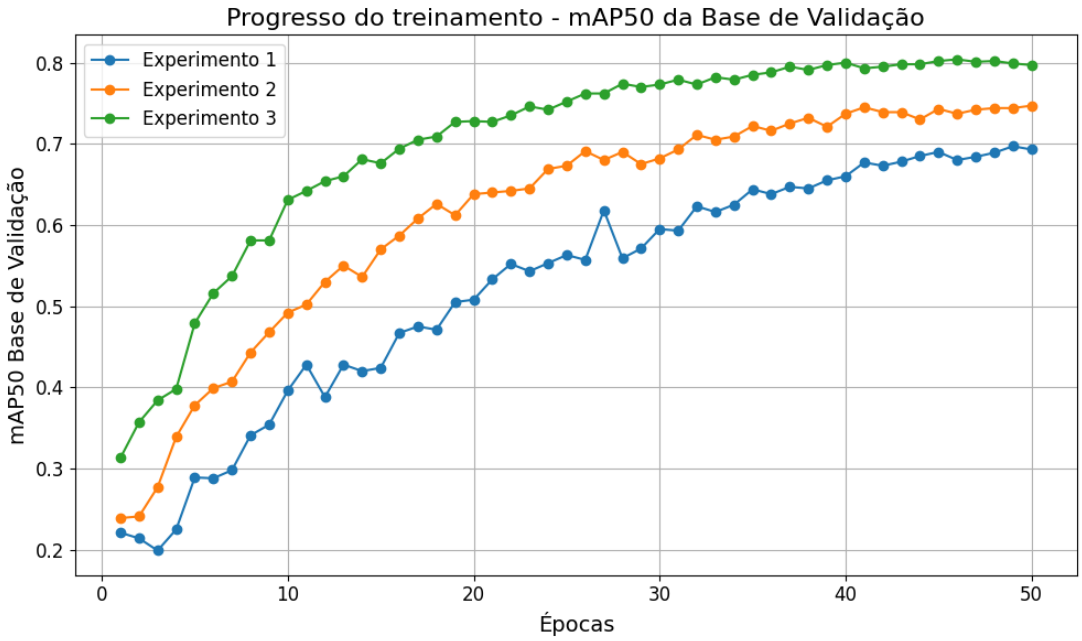

Figura 4.1: Progresso do treinamento

4.2 Análise de Desempenho

Para analisar o desempenho do modelo em cada classe, uma tabela com os mAP50 de cada classe é mostrada na Tabela 4.1. Pela Tabela 4.1, é possível verificar que as classes no geral melhoraram quando treinado com mais dados.

Tabela 4.1: Comparação do mAP50 de cada categoria nos 3 experimentos

Categoria	**Experimento 1**	**Experimento 2**	**Experimento 3**
short_sleeve_top	0.777	0.815	0.896
trousers	0.891	0.914	0.961
shorts	0.876	0.939	0.941
long_sleeve_top	0.643	0.762	0.795
skirt	0.685	0.813	0.848
vest_dress	0.338	0.479	0.644
short_sleeve_dress	0.657	0.732	0.818
vest	0.726	0.829	0.857
long_sleeve_outwear	0.784	0.764	0.823
long_sleeve_dress	0.575	0.682	0.833
sling_dress	0.652	0.694	0.817
sling	0.724	0.864	0.903
short_sleeve_outwear	0.714	0.868	0.916

4.3 Exemplos de Detecção

A partir daqui todos os exemplos mostrados usaram o modelo YOLO treinado no experimento 3, com a finalidade de demonstrar o uso do modelo que teve o melhor desempenho dentre os 3 experimentos realizados. A seguir, na Figura 4.2, um exemplo comparativo entre os objetos e suas *bounding boxes* detectadas pelo modelo comparado à referência verdadeira obtida a partir da base de dados.

Figura 4.2: Exemplo comparativo entre objetos detectados pela YOLO e a referência

4.4 Matrizes de Confusão

Para avaliar de forma mais precisa os casos específicos onde o modelo tende a apresentar um desempenho pior, foram construídas duas matrizes de confusão, uma normal e outra normalizada. A Figura 4.3 apresenta a matriz de confusão, na qual a classe `background` indica ausência de um objeto de interesse. Por exemplo, na classe prevista `background` com a classe verdadeira `short_sleeve_top` indica a quantidade de objetos da classe `short_sleeve_top` que estavam presentes na referência, mas não foram detectados pelo modelo. É importante ressaltar que a base de dados empregada para construir ambas matrizes de confusão é a seção de testes da base de dados utilizada no treinamento do experimento 3, utilizando o modelo de inferência do mesmo experimento.

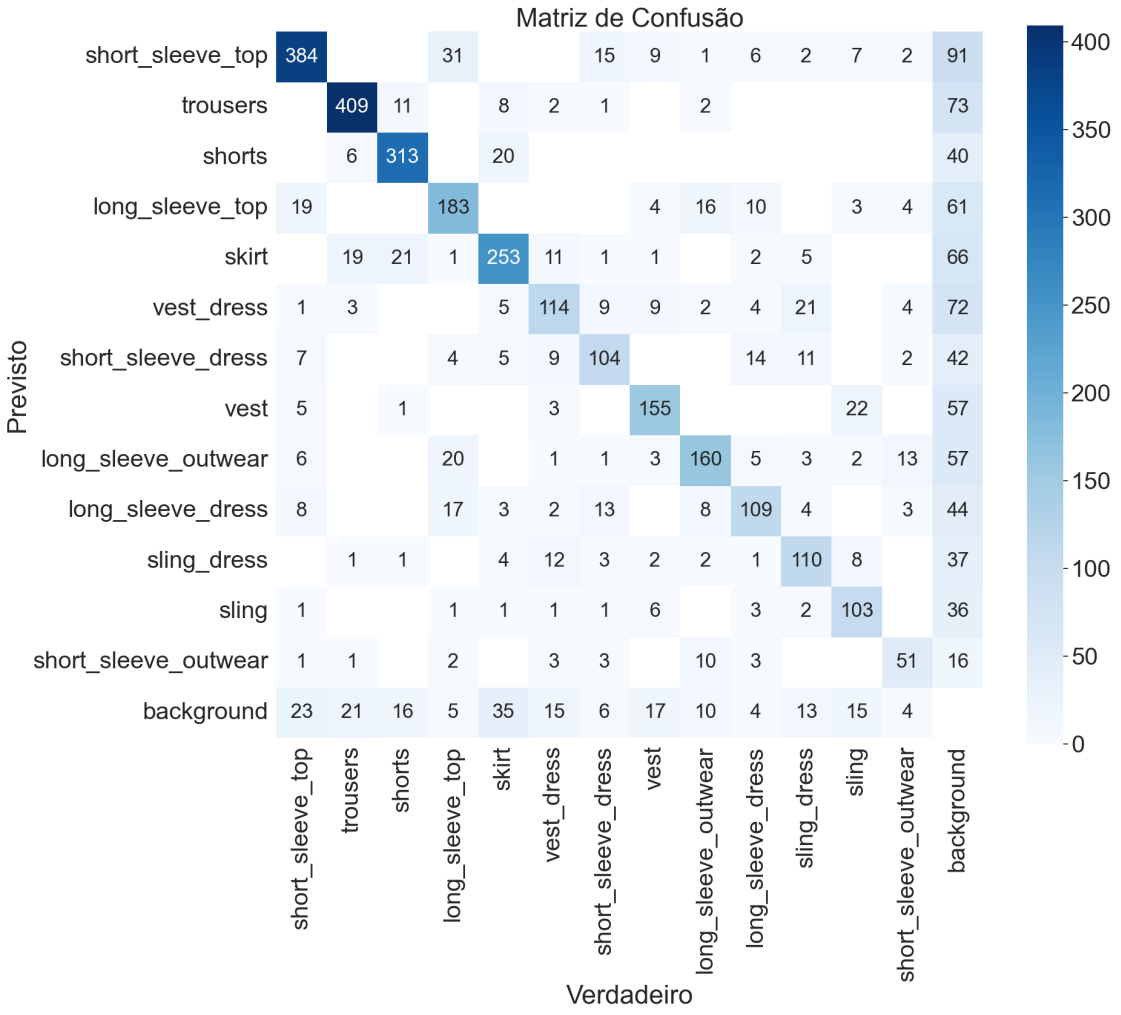

Figura 4.3: Matriz de confusão

A matriz de confusão normalizada presente na Figura 4.4 foi obtida ao dividir cada valor i, j da matriz pela soma de todos os objetos de cada classe verdadeira. Ou seja, para cada classe verdadeira representada no eixo horizontal (eixo j), o valor em i, j é dividido pela soma de todos os valores na coluna j.

A expressão matemática que representa esse cálculo é dada por:

$$\text{Matriz Normalizada}[i,j] = \frac{\text{Matriz de Confusão}[i,j]}{\sum_{i=1}^{N} \text{Matriz de Confusão}[i,j]}$$

onde N é o número total de classes.

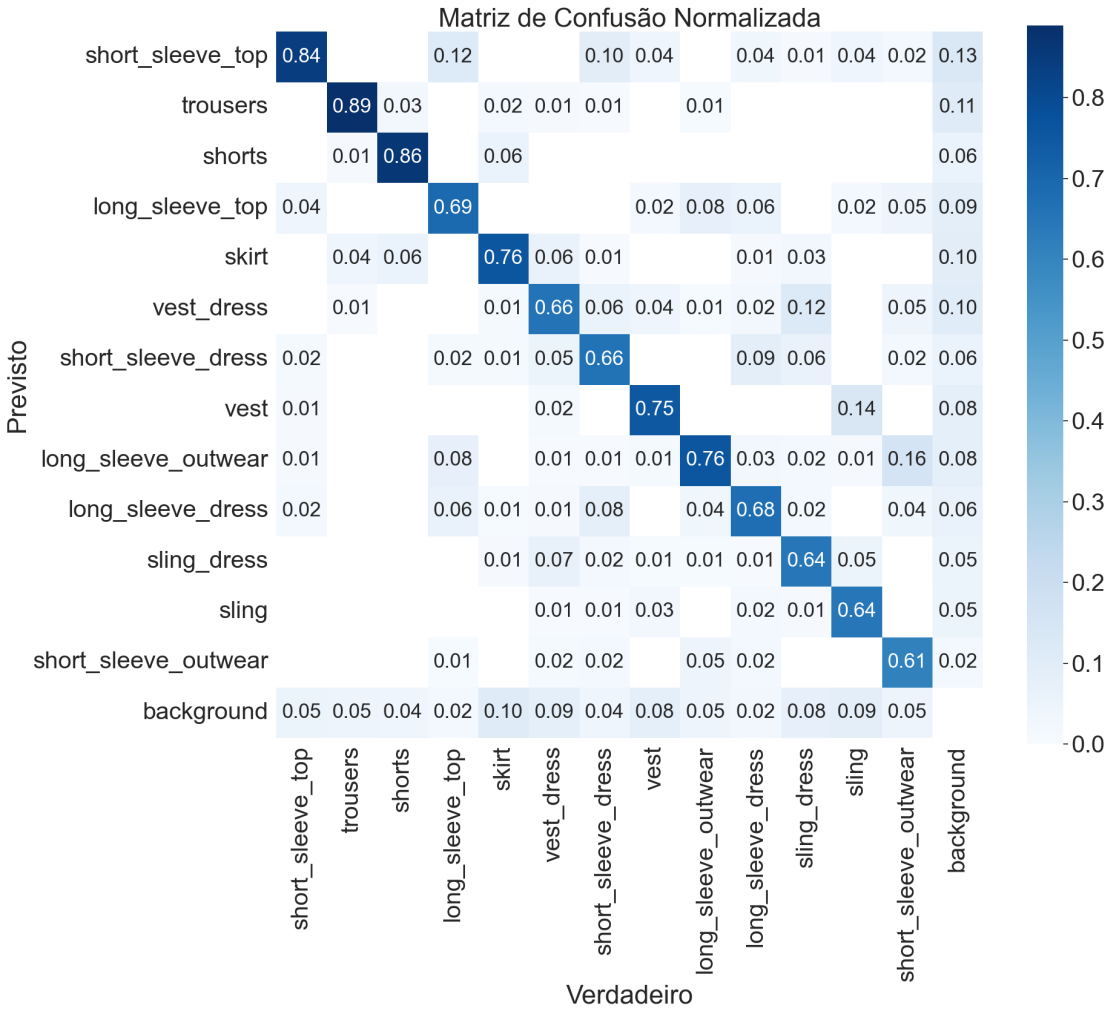

Figura 4.4: Matriz de confusão normalizada

4.5 Análise de Erros

De acordo com a matriz de confusão normalizada presente na Figura 4.4, os 3 casos com maiores erros foram:

1. Categoria verdadeira `short_sleeve_outwear` predito como `long_sleeve_outwear` com valor 0.16
2. Categoria verdadeira `sling` predito como `vest` com valor 0.14
3. Categoria verdadeira `background` predito como `short_sleeve_top` com valor 0.13

Com esses 3 pares de categorias preditas e verdadeiras, foram analisados exemplos comparativos das imagens e das *bounding boxes* com a finalidade de visualizar as imagens nas quais o modelo tende a performar pior. Na Figura 4.5 é demonstrado um caso onde

`short_sleeve_outwear` foi falsamente detectado como `long_sleeve_outwear`, provavelmente devido à manga estar de tamanho intermediário, podendo ser tanto manga longa como manga curta.

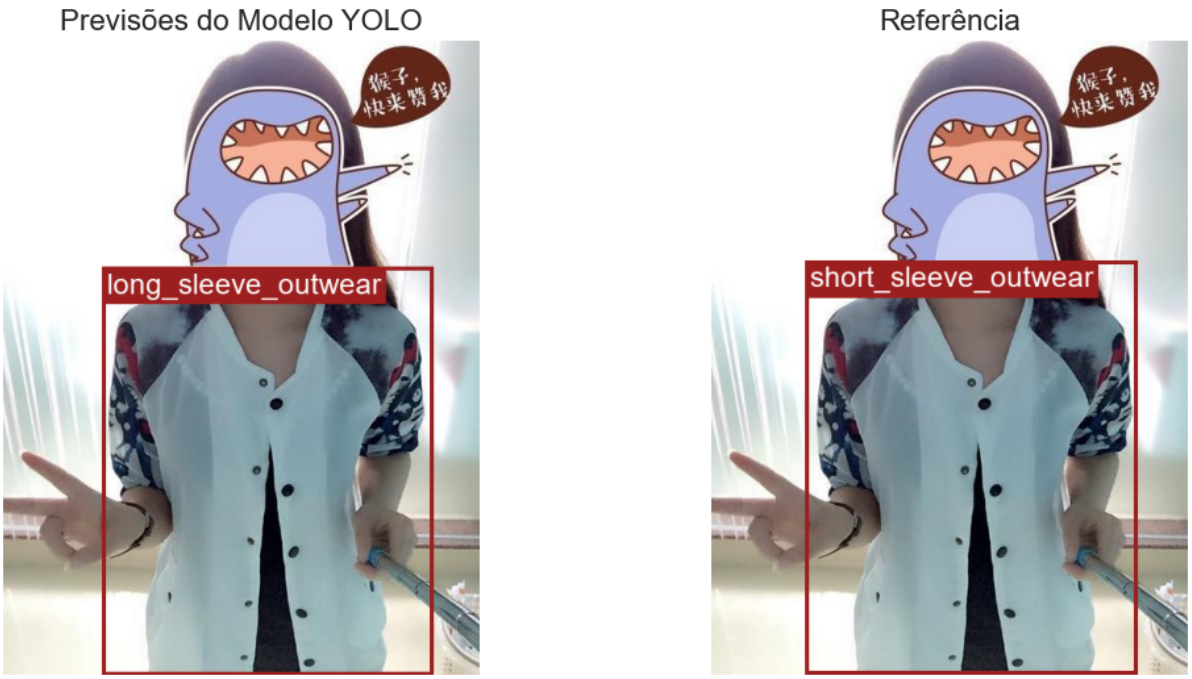

Figura 4.5: Exemplo de erro: Categoria verdadeira `short_sleeve_outwear` predito como `long_sleeve_outwear`

Na Figura 4.6, o objeto `sling` foi falsamente detectado como `vest`, provavelmente devido à similaridade entre as duas categorias.

Figura 4.6: Exemplo de erro: Categoria verdadeira `sling` predito como `vest`

Na Figura 4.7, o objeto `short_sleeve_top` foi predito como falso positivo, não estando presente na referência. Neste caso específico, devido ao modelo equivocadamente detectar a presença de 2 peças de roupas (`skirt` e `short_sleeve_top`) que fazem parte de uma única peça de roupa `vest_dress`.

Figura 4.7: Exemplo de erro: Categoria verdadeira `background` predito como `short_sleeve_top`

4.6 Busca por Similaridade

Para demonstrar o funcionamento do sistema completo com a busca por similaridade foram geradas as seguintes figuras:

Na Figura 4.8, é demonstrado um exemplo de funcionamento do sistema completo com a imagem de busca com a *bounding box* do tipo `vest_dress` detectada. Em seguida, um recorte é feito do objeto de interesse e a busca por similaridade foi realizada usando distância euclidiana.

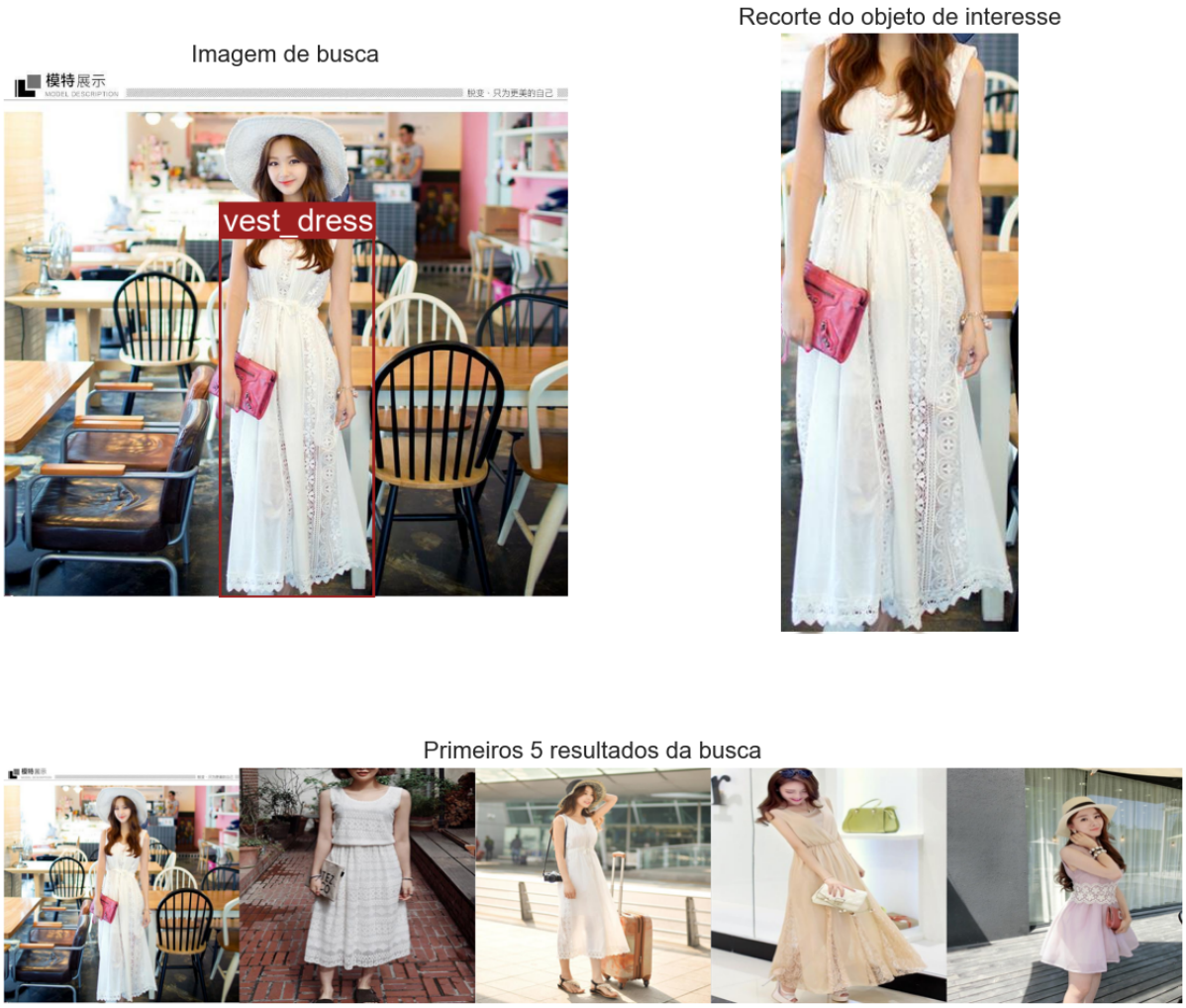

Figura 4.8: Exemplo da busca pelo objeto `vest_dress`

Figura 4.9: Exemplo da busca pelo objeto `trousers`

Na Figura 4.9, está presente um outro exemplo de busca, dessa vez pelo objeto `trousers` e na Figura 4.10 um novo exemplo de busca usando a mesma imagem, porém dessa vez pelo objeto `sling`.

Figura 4.10: Exemplo de busca pelo objeto `sling`

4.7 Avaliação da Busca por Similaridade

Foi realizada uma avaliação da etapa de busca por similaridade, utilizando duas diferentes categorias: `short_sleeve_top` e `trousers`. Para cada categoria, uma imagem aleatória foi selecionada como imagem de consulta e outras 50 imagens foram selecionadas aleatoriamente como galeria, representando o conjunto de dados a ser pesquisado.

A busca foi realizada utilizando os embeddings gerados para as imagens da galeria e comparados com o embedding da imagem de consulta. Os itens relevantes foram definidos manualmente, com base na similaridade visual percebida. Para avaliar o desempenho do sistema de busca, foram utilizadas as seguintes métricas:

- **Precisão@k**: a fração dos k primeiros itens retornados que são relevantes.
- **Cobertura@k** (ou *Recall@k*): a fração de itens relevantes que foram recuperados nos k primeiros itens.
- **F1-Score@k**: a média harmônica entre a Precisão e a Cobertura.

Foram realizados dois experimentos, um para cada categoria de roupa, utilizando o valor de $k = 3$ para a categoria `short_sleeve_top` e $k = 3$ para a categoria `trousers`.

4.7.1 Experimento 1: Short Sleeve Top

Para a categoria `short_sleeve_top`, obtivemos os seguintes resultados:

- **Precisão@3**: 0.67
- **Cobertura@3**: 0.67
- **F1-Score@3**: 0.67

Figura 4.11: Experimento 1: exemplo da busca pelo objeto `short_sleeve_top`

4.7.2 Experimento 2: Trousers

Para a categoria `trousers`, obtivemos os seguintes resultados:

- **Precisão@3**: 0.67
- **Cobertura@3**: 0.67

- **F1-Score@3**: 0.67

Figura 4.12: Experimento 2: exemplo da busca pelo objeto `trousers`

Os resultados indicam que o sistema apresentou um desempenho razoável na tarefa de busca por similaridade de imagens. Para futuras melhorias, o ideal seria realizar um *fine-tuning* do extrator de características DinoV2.

5

Conclusões e Trabalhos Futuros

Este trabalho apresentou o desenvolvimento de um sistema de busca por similaridade para peças de vestuário utilizando técnicas de detecção de objetos, extração de características e busca por vizinhos mais próximos. O trabalho contou com a análise da base de dados DeepFashion2, o treinamento do modelo de detecção usando YOLOv11 em conjunto com a análise dos resultados do treinamento, e a implementação de um sistema de busca por similaridade visual usando os *embeddings* gerados por um modelo pré-treinado DINOv2 e a distância euclidiana para o cálculo de similaridade.

O detector de objetos obteve bons resultados na métrica de mAP50, principalmente no experimento 3, que contém mais objetos de treino. Porém, os resultados para certas classes, como `vest_dress`, ficaram atrás no experimento 1, mas melhoraram no experimento 3, demonstrando que certas classes, provavelmente por serem mais difíceis para o modelo, precisam de mais exemplos na base de dados para se obter uma melhor performance. Enquanto isso, outras classes, como `trousers`, não tiveram melhorias significativas entre os experimentos. Na análise da matriz de confusão, foi notado que o modelo tem mais dificuldade de diferenciar classes visualmente muito similares, como entre `vest_dress` e `sling_dress`, o que poderia ser resolvido juntando classes similares em uma única classe. Dependendo da aplicação de um sistema como esse, essa junção de classes similares não seria um problema.

De acordo com as informações obtidas por este trabalho de forma experimental, concluiu-se que este sistema poderia ser utilizado sem muitos problemas por pequenas organizações que têm poucos recursos, visto que o tempo de treinamento necessário foi de cerca de apenas 7 horas utilizando uma GPU intermediária de consumo individual, graças ao uso de modelos pré-treinados *open source* como DINOv2 e YOLOv11. Além disso, considerando que a base de dados DeepFashion2 tem uma licença que proíbe o uso para fins comerciais e, de acordo com os experimentos 1, 2 e 3, cerca de apenas 1000 objetos anotados por classe já são suficientes para resultados satisfatórios para uso comercial. Com a junção de classes e o

uso de técnicas de aumento sintético de dados, seria possível reduzir ainda mais o tamanho da base de dados necessário para o treino.

Em trabalhos futuros, é necessário um *fine-tuning* do extrator de características para melhorar a separação no espaço latente das características visuais extraídas pelo DINOv2, facilitando um agrupamento melhor das diferentes peças de vestuário com base em suas semelhanças visuais. Como a intenção deste projeto é criar um sistema viável para pequenas organizações, muitas vezes há a necessidade de treinar o modelo de detecção de objetos usando uma base de dados própria. Para isso, é necessário minimizar o tamanho da base de treinamento necessária devido ao custo de anotação da mesma. Em trabalhos futuros, seria interessante a experimentação de técnicas de aumento sintético dos dados de treino para diminuir ainda mais o tamanho da base de dados necessária para o treino.

Apêndice A

Documentação do Software

A.1 Instalação

Para instalar o sistema, é necessário ter o Python instalado. A seguir, são apresentadas as instruções de instalação para sistemas Windows e Linux.

A.1.1 Windows

1. Abra o Prompt de Comando (cmd) e navegue até o diretório do projeto.
2. Crie um ambiente virtual:

```
python -m venv .venv
```

3. Ative o ambiente virtual:

```
.venv\Scripts\Activate
```

4. Instale as dependências do projeto:

```
pip install -r requirements.txt
```

A.1.2 Linux

1. Abra o terminal e navegue até o diretório do projeto.
2. Crie um ambiente virtual:

```
python3 -m venv .venv
```

3. Ative o ambiente virtual:

```
source .venv/bin/activate
```

4. Instale as dependências do projeto:

```
pip install -r requirements.txt
```

A.2 Jupyter Notebooks Implementados

Três Jupyter notebooks foram implementados:

- `maps.ipynb`: Mostra os mAPs dos experimentos.
- `deepfashion2_data_analysis.ipynb`: Mostra as análises estatísticas da base de dados DeepFashion2.
- `result.ipynb`: Mostra os gráficos com resultados dos experimentos, juntamente com matrizes de confusão e alguns experimentos demonstrando o funcionamento do sistema.

A.3 Arquivos Implementados

- `train.py`: Responsável pelo treino do modelo YOLO. É onde podem ser modificados os hiperparâmetros do treino. É esperado que, ao ser executado, os dados de treino estejam no formato correto na pasta `datasets`, de acordo com o padrão que o YOLO requer.
- `create_df_experiment.py`: Responsável por criar a base de dados que será usada para treinar o modelo YOLO, utilizando uma amostragem da base DeepFashion2 e fazendo as divisões de treino, teste e validação necessárias para os três experimentos.
- `create_embeddings.py`: Responsável por criar os embeddings de todos os objetos presentes na seção de treino da base de dados DeepFashion2. Esses embeddings são armazenados em arquivos `npz` com o nome da categoria à qual pertencem.
- `deepfashion2_to_yolo.py`: Responsável por construir a base de dados de treino no formato YOLO a partir das imagens e anotações presentes na pasta de treino da DeepFashion2, usando os itens selecionados pelo arquivo `create_df_experiment.py`.

Referências bibliográficas

Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Yuying Ge, Ruimao Zhang, Lingyun Wu, Xiaogang Wang, Xiaoou Tang, and Ping Luo. A versatile benchmark for detection, pose estimation, segmentation and re-identification of clothing images. *CVPR*, 2019.

Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):142–158, 2015.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Yushi Jing, David Liu, Dmitry Kislyuk, Andrew Zhai, Jiajing Xu, Jeff Donahue, and Sarah Tavel. Visual search at pinterest. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1889–1898, 2015.

Glenn Jocher and Jing Qiu. Ultralytics yolo11, 2024. URL https://github.com/ultralytics/ultralytics.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Michal Kucer and Naila Murray. A detect-then-retrieve model for multi-domain fashion item retrieval. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 0–0, 2019.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2010.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Jacob Solawetz and Francesco. What is yolov8? a complete guide. URL https://blog.roboflow.com/what-is-yolov8/.