UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Miguel Luiz Pessoa da Cruz Silva

Detecção de Objetos com Saídas Antecipadas Baseadas em Complexidade da Imagem

Recife

2025

Miguel Luiz Pessoa da Cruz Silva

Detecção de Objetos com Saídas Antecipadas Baseadas em Complexidade da Imagem

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação, Inteligência Computacional, Detecção de Objetos

Orientador (a): Tsang Ing Ren

Recife

2025

.Catalogação de Publicação na Fonte. UFPE - Biblioteca Central

Silva, Miguel Luiz Pessoa da Cruz.
Detecção de objetos com saídas antecipadas baseadas em complexidade da imagem / Miguel Luiz Pessoa da Cruz Silva. - Recife, 2025.
123f.: il.

Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação/PPGCC.
Orientação: Tsang Ing Ren.

1. Redes Neurais Dinâmicas; 2. Detecção de Objetos; 3. Saídas Antecipadas. I. Ren, Tsang Ing. II. Título.

UFPE-Centro de Ciencias Exatas e da Natureza

Miguel Luiz Pessoa da Cruz Silva

“Detecção de Objetos com Saídas Antecipadas Baseadas em Complexidade da Imagem”

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Inteligência Computacional

Aprovado em: 19/02/2025.

BANCA EXAMINADORA

_____

Prof. Dr. George Darmiton da Cunha Cavalcanti
Centro de Informática / UFPE

_____

Prof. Dr. Alceu de Souza Britto Junior
Programa de Pós Graduação em Informática Aplicada/PUC-PR

_____

Prof. Dr. Tsang Ing Ren
Centro de Informática
(**orientador**)

AGRADECIMENTOS

Gostaria de expressar minha profunda gratidão a todos que tornaram possível a realização desta dissertação.

Um agradecimento carinhoso aos meus pais, Luiz e Rosangela, que sempre reforçaram o valor do conhecimento, do estudo e do trabalho duro. A minha duplinha, Nicole, que nunca me deixou duvidar de que eu era capaz. Aos meus avós, Izidio, Teca e Amara, pelo eterno porto seguro. As minhas irmãs, Laura e Bruna, pelo apoio em momentos difíceis.

Agradeço também aos amigos que acompanharam essa jornada, e em especial, Lama, Pontes e Leilane, que me ajudaram a enxergar caminhos e possibilidades para realização, através de suas revisões e sugestões.

Um agradecimento especial é dirigido ao meu orientador, Professor Tsang, cuja paciência e apoio foram fundamentais durante todo o processo de pesquisa e escrita.

RESUMO

Redes Neurais Dinâmicas são modelos de aprendizado profundo que ajustam sua complexidade computacional durante a inferência, adaptando o uso de recursos de acordo com a dificuldade da entrada. Dentre essas abordagens, a técnica de saídas antecipadas (*early exit*) se destaca por permitir que o modelo interrompa a inferência antecipadamente em amostras mais simples, reduzindo o tempo de processamento e o consumo de recursos, sem comprometer significativamente a precisão. No entanto, sua aplicação em tarefas de detecção de objetos ainda é pouco explorada, especialmente devido à complexidade de lidar com múltiplos objetos e diferentes níveis de confiança em uma mesma imagem.

Este trabalho propõe uma nova abordagem ao adaptar a arquitetura *Single Shot MultiBox* Detector (SSD) com ramificações de saída antecipada, permitindo que amostras simples sejam processadas de forma mais eficiente enquanto entradas mais complexas recebem uma análise completa. Para isso, introduz-se um modelo auxiliar para predição do nível de dificuldade, baseado em uma metodologia de classificação de complexidade de amostras de uma base de dados de detecção. Embora a Precisão Média (mAP) obtida pela abordagem proposta permaneça comparável ao modelo *single-exit*, os resultados experimentais demonstram uma redução significativa no tempo de inferência e no uso de recursos computacionais para o subconjunto de imagens mais simples, evidenciando a eficiência da estratégia para aplicações em tempo real, especialmente em cenários com restrições computacionais.

Palavras-chaves: Early Exit, Redes Neurais dinâmicas, detecção de objetos.

ABSTRACT

Dynamic Neural Networks are deep learning models that adjust their computational complexity during inference, adapting resource usage according to the input difficulty. Among these approaches, the *early exit* technique stands out, as it allows the model to prematurely interrupt inference on simpler samples, thus reducing processing time and resource consumption without significantly compromising accuracy. However, its application in object detection tasks remains underexplored, especially due to the complexity associated with handling multiple objects and varying confidence levels within a single image.

This work proposes a novel approach by adapting the *Single Shot MultiBox Detector* (SSD) architecture with *early exit* branches, allowing simpler samples to be processed more efficiently while ensuring a comprehensive analysis for more complex inputs. To this end, we introduce an auxiliary model for predicting the difficulty level, based on a methodology for classifying sample complexity within an object detection dataset. Although the Mean Average Precision (mAP) obtained by the proposed approach remains comparable to the *single-exit* model, experimental results demonstrate a significant reduction in inference time and computational resource usage for the simpler subset of images. This underscores the efficiency of the strategy for real-time applications, particularly in computationally constrained scenarios.

Keywords: Early Exit, Dynamic Neural Networks, Object Detection

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O sistema visual humano é um mecanismo complexo e altamente desenvolvido de percepção sensorial, responsável por tarefas como identificação de padrões, reconhecimento de movimento e detecção de objetos (BIEDERMAN, 1987). Esse sistema evoluiu ao longo de milhões de anos, otimizando-se para interpretar uma ampla gama de estímulos visuais encontrados no ambiente. Além de ser essencial para a sobrevivência em ambientes predatórios, a visão desempenhou um papel fundamental no desenvolvimento da linguagem, desde representações gráficas em cavernas até a evolução da escrita (BEDNY; RICHARDSON; SAXE, 2015).

Com os avanços nas tecnologias digitais, a busca por automatizar processos relacionados à percepção visual levou ao surgimento da visão computacional, um campo dedicado à análise e interpretação de estímulos visuais digitais. Os trabalhos pioneiros nessa área, como os de Roberts (ROBERTS, 1963) e Papert (PAPERT, 1966), estabeleceram as bases do campo, que desde então têm evoluído significativamente.

Os avanços em processamento de imagem desempenharam um papel central no progresso da visão computacional, permitindo o desenvolvimento de algoritmos mais precisos e eficientes. A combinação dessas técnicas viabilizou aplicações em tarefas como classificação de objetos (SHEN, 2019), segmentação de imagens (HAFIZ; BHAT, 2020) e detecção de objetos (ZAIDI et al., 2021), utilizando o conceito de *extratores de características* (SALAU; JAIN, 2020).

A detecção de objetos, em particular, envolve identificar e localizar objetos dentro de uma imagem ou vídeo, atribuindo a cada objeto um rótulo de classe e delimitando sua posição por meio de *bounding boxes* (ZAIDI et al., 2021). Os avanços na detecção de objetos têm sido impulsionados pelo uso de redes neurais profundas, que superaram as limitações de métodos baseados em características manuais.

Apesar do conceito de redes neurais ter sido introduzido nos anos 1940 (MCCULLOCH; PITTS, 1943), foi apenas no final dos anos 1950, com a proposta do *Perceptron* (ROSENBLATT, 1958), que se começou a explorar seu potencial em tarefas de aprendizado de máquina. Desde então, as redes neurais têm evoluído para modelos mais complexos, inspirados na maneira como os humanos aprendem a realizar tarefas, identificando padrões em vez de aplicar regras pré-definidas.

Nos anos 2010, redes convolucionais profundas (*Convolutional Neural Networks*, CNNs) revolucionaram a visão computacional. Trabalhos como AlexNet (KRIZHEVSKY; SUTSKEVER;

HINTON, 2012a) e Faster R-CNN (REN et al., 2015) demonstraram como redes treináveis de ponta a ponta poderiam integrar classificação e detecção em um único modelo, alcançando alta precisão em cenários desafiadores. Posteriormente, o *Single Shot MultiBox Detector* (SSD) (LIU et al., 2016) foi introduzido como uma arquitetura que melhora a eficiência computacional na detecção de objetos ao realizar predições em múltiplas escalas simultaneamente, sem a necessidade de uma etapa de proposta de regiões separada. O SSD alcança um equilíbrio notável entre velocidade e precisão, tornando-o uma escolha popular para aplicações em tempo real e dispositivos com recursos limitados.

Diversas abordagens estáticas têm sido amplamente exploradas para reduzir o tempo de inferência e o uso de recursos computacionais em redes neurais profundas, incluindo técnicas como *pruning*, destilação de conhecimento (*knowledge distillation*) e compressão de modelos. O *pruning* visa remover conexões ou neurônios redundantes, reduzindo diretamente a complexidade da rede sem perdas significativas de desempenho (HAN; MAO; DALLY, 2016). A destilação de conhecimento permite transferir o desempenho de modelos maiores para modelos menores, otimizando a eficiência computacional sem comprometer drasticamente a precisão (HINTON; VINYALS; DEAN, 2015). Já a compressão de modelos envolve técnicas como quantização e decomposição de tensores, que reduzem a precisão dos parâmetros ou decompõem operações complexas para tornar o modelo mais eficiente durante a execução (CHENG et al., 2017). Tais abordagens são consideradas estáticas porque, após a etapa de otimização ou compressão, a arquitetura do modelo permanece fixa durante a inferência, não sendo capaz de adaptar dinamicamente o seu processamento às características específicas das entradas individuais.

Em contraste com as técnicas estáticas, abordagens dinâmicas ajustam o processamento em tempo real, adaptando-se às características específicas das entradas durante a inferência. Entre essas abordagens destacam-se as redes neurais com saídas antecipadas (*early exits*), que permitem interromper o processamento precocemente em amostras mais simples, economizando tempo e recursos computacionais (TEERAPITTAYANON; MCDANEL; KUNG, 2016). Além disso, estratégias como atenção seletiva (*selective attention*) possibilitam que o modelo concentre esforços em regiões mais relevantes das entradas, evitando desperdício computacional em áreas menos importantes (MNIH; HEESS; GRAVES, 2014). Técnicas como inferência adaptativa (*adaptive inference*) também podem ajustar dinamicamente o número de camadas ou operações executadas, dependendo da complexidade percebida de cada amostra (FIGURNOV et al., 2017). Tais abordagens dinâmicas destacam-se por sua capacidade de flexibilizar o processamento, maximizando a eficiência sem comprometer significativamente a precisão do

modelo.

No contexto de detecção de objetos, modelos dinâmicos, como o AdaDet (YANG et al., 2023), introduziram saídas antecipadas (*early exits*), que permitem encerrar a inferência em camadas intermediárias, adaptando o processamento com base na complexidade da entrada. Entretanto, essas arquiteturas ainda apresentam limitações, como o uso de limiares de confiança estáticos, que podem não capturar a real dificuldade da entrada.

Essas abordagens não incorporam mecanismos para avaliar de maneira robusta a dificuldade da entrada em tempo real, o que pode limitar a capacidade dos modelos dinâmicos de se adaptarem de forma eficaz a diferentes níveis de complexidade. Trabalhos recentes, como os de (LIN et al., 2023), tentam abordar essas questões introduzindo estratégias mais sofisticadas para a determinação de saídas antecipadas. No entanto, ainda há espaço para melhorias que permitam que essas decisões sejam baseadas em uma análise mais profunda e contextual das entradas, por exemplo, através da utilização de métricas adaptativas de dificuldade, integração de mecanismos de atenção, ou adoção de políticas de decisão dinâmicas que considerem tanto a confiança das saídas quanto a complexidade intrínseca dos dados de entrada.

Apesar dos avanços significativos nas técnicas de otimização estática e dinâmica para redes neurais profundas, ainda existe uma lacuna importante no contexto de detecção de objetos. Abordagens dinâmicas, como saídas antecipadas, têm se mostrado eficazes principalmente em tarefas de classificação, mas sua aplicação em detecção de objetos permanece limitada devido à complexidade intrínseca da tarefa, que envolve múltiplas previsões simultâneas com diferentes níveis de confiança em uma única imagem. Além disso, métodos existentes frequentemente dependem de limiares fixos ou heurísticas que não se adaptam bem à variabilidade natural das entradas. Nesse cenário, torna-se necessário o desenvolvimento de um novo algoritmo que não apenas integre saídas antecipadas de forma eficaz em arquiteturas de detecção, mas também seja capaz de avaliar dinamicamente a dificuldade de cada amostra, garantindo decisões mais informadas e um balanceamento robusto entre eficiência e precisão. Tal proposta visa preencher a lacuna atual, ampliando o potencial de uso de modelos dinâmicos em aplicações de detecção em tempo real, especialmente em cenários restritos por recursos computacionais.

Este trabalho visa abordar essas lacunas, propondo uma metodologia que integra saídas antecipadas a um modelo de detecção base, o SSD (LIU et al., 2016), modificado com uma saída antecipada no *backbone* ResNet101, e uma política de decisão baseada na estimativa da dificuldade da entrada. Essa abordagem busca equilibrar eficiência computacional e precisão, oferecendo uma solução adaptável para cenários diversos.

Os resultados quantitativos demonstraram uma redução significativa no uso de memória GPU em $16,75\%$, uma diminuição no tempo médio de inferência em $11,10\%$ e uma redução no consumo energético em $14,09\%$, com uma leve redução de apenas 1% na precisão média (mAP) comparado ao modelo original. Tais resultados evidenciam o potencial da estratégia proposta, especialmente em aplicações em tempo real e contextos com restrições computacionais.

1.1 MOTIVAÇÃO

A crescente demanda por sistemas eficientes em tempo real, como veículos autônomos e dispositivos embarcados, exige modelos que equilibrem precisão e eficiência computacional. Redes dinâmicas, especialmente aquelas com saídas antecipadas, apresentam um grande potencial para atender a esses requisitos. Apesar de amplamente exploradas em tarefas de classificação, sua aplicação na detecção de objetos é limitada, representando um campo promissor para pesquisa.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Desenvolver uma arquitetura dinâmica para detecção de objetos, integrando saídas antecipadas com uma política adaptativa baseada na estimativa de dificuldade da entrada.

1.2.2 Objetivos Específicos

- Propor uma métrica para quantificar a dificuldade das entradas;
- Desenvolver um modelo para estimar a dificuldade durante a inferência;
- Projetar uma política de saída adaptativa baseada na dificuldade estimada;
- Adaptar a arquitetura SSD para incorporar saídas antecipadas;
- Avaliar o desempenho do modelo proposto em comparação com arquiteturas convencionais.

CONTRIBUIÇÕES

As principais contribuições deste trabalho são:

- Extensão do paradigma de *early-exit neural networks* para a detecção de objetos;
- Desenvolvimento de uma métrica de dificuldade para entradas;
- Análise das estratégias de treinamento e políticas de saída para redes dinâmicas;
- Adaptação detalhada da arquitetura SSD para suportar saídas antecipadas;
- Avaliação experimental comparativa, destacando ganhos em eficiência computacional e precisão.

ESTRUTURA DA DISSERTAÇÃO

Os capítulos desta dissertação estão organizados da seguinte forma:

- **Capítulo 2:** Apresentação dos conceitos fundamentais para o entendimento do trabalho.
- **Capítulo 3:** Revisão do campo de redes neurais dinâmicas aplicadas à detecção de objetos.
- **Capítulo 4:** Descrição detalhada do método proposto neste trabalho.
- **Capítulo 5:** Apresentação dos experimentos realizados e os resultados obtidos.
- **Capítulo 6:** Conclusões a respeito do trabalho e discussões sobre direções futuras.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 INTRODUÇÃO

A compreensão dos conceitos fundamentais de redes neurais artificiais e suas variações é essencial para o desenvolvimento de modelos eficazes na tarefa de detecção de objetos. Este capítulo apresenta uma revisão detalhada dos principais conceitos, estruturas e avanços das áreas de estudo desse trabalho.

Inicialmente, serão discutidos os princípios das redes neurais artificiais, abordando suas estruturas básicas, as diferentes camadas que as compõem e os operadores utilizados para facilitar o aprendizado de máquina. Em seguida, será explorado o funcionamento das redes neurais convolucionais (CNNs), destacando sua estrutura, operação e as inovações recentes que têm impulsionado seu desempenho em várias aplicações de visão computacional.

O capítulo também examinará as redes neurais dinâmicas, uma classe de modelos que adapta sua arquitetura e comportamento com base na complexidade das entradas e nos requisitos de saída. Serão apresentados os diferentes tipos de redes dinâmicas, ressaltando as vantagens e desvantagens de cada abordagem.

Finalmente, será introduzida a base teórica da detecção de objetos, uma tarefa crucial em visão computacional. A seção cobrirá as abordagens tradicionais e modernas para detecção de objetos, os desafios enfrentados por esses métodos e as métricas utilizadas para avaliar seu desempenho. A compreensão desses conceitos fornecerá a base necessária para a análise crítica e o desenvolvimento da arquitetura proposta neste trabalho.

2.2 REDES NEURAIS

2.2.1 Redes Neurais Artificiais

As Redes Neurais Artificiais (ANN) constituem um conjunto de técnicas de inteligência artificial inspiradas na estrutura e no funcionamento do cérebro humano, caracterizando-se pela capacidade de aprender a partir de padrões de dados por meio de redes de neurônios interconectados. Essas arquiteturas têm-se mostrado eficazes em inúmeras áreas, como visão computacional (por exemplo, reconhecimento de objetos em imagens), processamento de linguagem natural (como tradução automática) e sistemas de recomendação (personalização de

conteúdo).

O conceito bioinspirado de Neurônio Artificial foi introduzido na década de 1940, quando Warren McCulloch e Walter Pitts propuseram um modelo matemático para neurônios artificiais, inspirado no sistema nervoso biológico (MCCULLOCH; PITTS, 1943). A analogia entre neurônios biológicos e artificiais se baseia no conceito de unidades interconectadas, que recebem informações dos seus vizinhos, combinando-as em um novo estímulo e o propagando, formando, assim, uma rede. Nos neurônios biológicos, os estímulos são de natureza elétrica e química, e se comunicam através de sinapses que, ao atingirem um neurônio, são integradas por processos bioquímicos, podendo ser lineares ou não, a depender de fatores biológicos. Após essa integração, há um mecanismo de atenuação do estímulo a ser propagado, baseado em se a combinação das entradas excede ou não um certo limiar. De forma simplificada, o processo de aprendizado biológico envolve ajustar a força entre as conexões dos neurônios, ditando a influência das sinapses (KANDEL; SCHWARTZ; JESSELL, 2000).

Na modelagem matemática, o conjunto das forças será chamado de *pesos*, a modulação de saída, de *função de ativação*, e o processo de ajuste dos pesos, de *treinamento*.

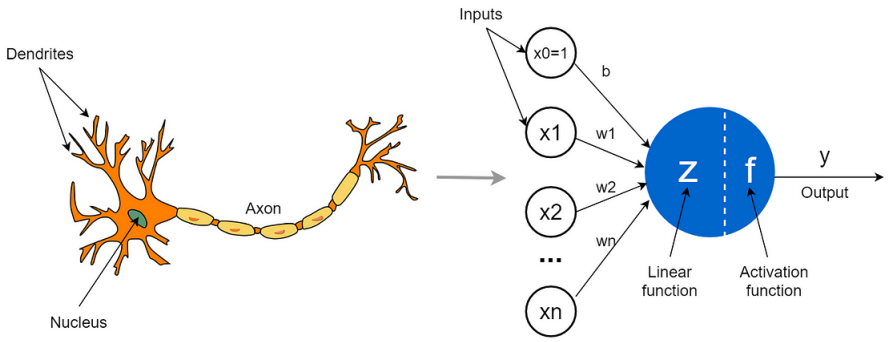


Figura 1 – Paralelo entre neurônio biológico e artificial. (Towards Data Science, 2024)

Um dos primeiros modelos de neurônio artificial é o *perceptron* (ROSENBLATT, 1958), introduzido por Frank Rosenblatt em 1958. O *perceptron* é a unidade básica das redes neurais artificiais e pode ser visto como um classificador linear. Ele recebe várias entradas, aplica um peso a cada uma e calcula uma soma ponderada dessas entradas. Em seguida, essa soma é passada por uma função de ativação, geralmente uma função degrau, que decide se o neurônio será ativado (saída 1) ou não (saída 0). A regra de atualização dos pesos do *perceptron* durante o treinamento é baseada no erro entre a saída prevista e a saída desejada, ajustando os pesos de forma iterativa para minimizar esse erro.

No entanto, a limitação do *perceptron* clássico em resolver problemas não linearmente separáveis freou o progresso na área por algum tempo. Demonstrou-se, por exemplo, que o *perceptron* de camada única não era capaz de resolver problemas como o XOR, o que levou a um período de estagnação no campo, conhecido como *"Inverno da IA"* (MINSKY; PAPERT, 1969). Durante essa fase, a comunidade científica voltou seus esforços para outras abordagens, como sistemas baseados em regras e métodos estatísticos, que pareciam promissores diante das dificuldades enfrentadas pelas redes neurais.

Apesar disso, alguns pesquisadores continuaram a explorar possibilidades para superar essas limitações. A questão central era encontrar maneiras de estender a capacidade de aprendizado das redes neurais, permitindo a solução de problemas que dependiam de maior complexidade nas representações, e o ressurgimento do interesse em redes neurais veio em meados dos anos 1980, com a introdução de novos modelos e algoritmos, como a retropropagação do erro (*back-propagation*), apresentada por Rumelhart, Hinton e Williams em 1986 (RUMELHART; HINTON; WILLIAMS, 1986). Esse algoritmo permitiu o treinamento eficiente de redes neurais multicamadas, também conhecidas como *Multi-Layer Perceptrons* (MLPs), superando as limitações do *perceptron* de camada única. A capacidade de introduzir profundidade, aliada a funções de ativação não lineares, possibilitou a aprendizagem de representações complexas e a solução de problemas não linearmente separáveis. Além disso, o Teorema da Aproximação Universal (HORNIK; STINCHCOMBE; WHITE, 1989) formaliza a capacidade de MLPs (com ao menos uma camada oculta) em aproximar, sob condições adequadas, qualquer função contínua, o que reforça o potencial de representação dessas arquiteturas.

A equação que descreve a saída de um *perceptron* pode ser representada da seguinte forma:

$$y = f(x) = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.1}$$

onde: y é a saída do neurônio, ϕ é a função de ativação, w_i são os pesos das entradas, x_i são as entradas, b é o termo de viés (*bias*) e n é o número de entradas do neurônio. Essa formulação segue a representação ilustrada na Figura 1.

Cada elemento do neurônio tem um papel específico na capacidade de representação do modelo. Fazendo uma analogia com uma equação linear, os pesos podem ser comparados aos coeficientes de inclinação de uma reta. Já o viés atua como um deslocamento, permitindo que a reta se mova para cima ou para baixo no gráfico. Isso possibilita que o neurônio modele retas que não passam necessariamente pela origem, aumentando a flexibilidade e a capacidade

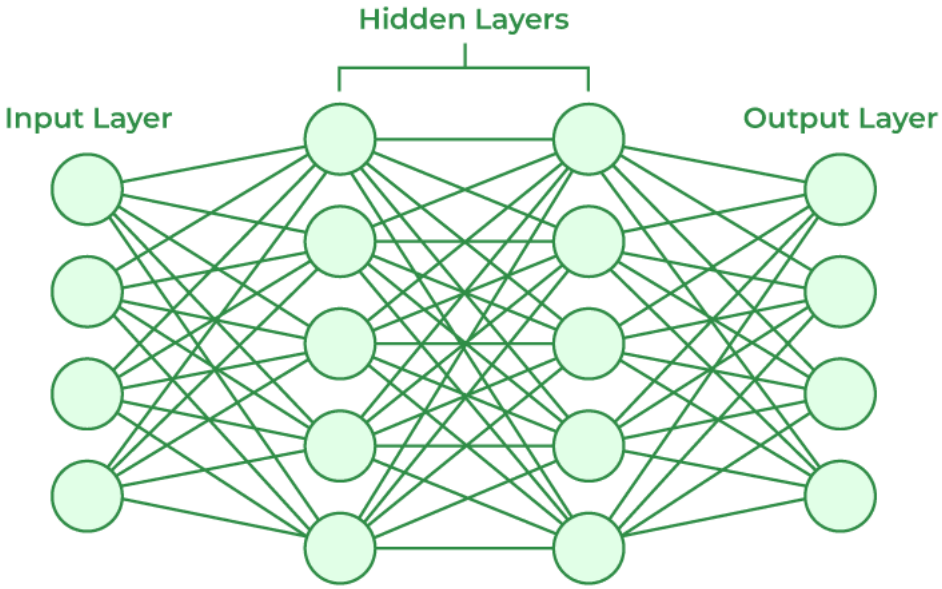


Figura 2 – Ilustração de uma MLP genérica (GeeksforGeeks, 2024)

do modelo de ajustar-se a diferentes conjuntos de dados (GOODFELLOW; BENGIO; COURVILLE, 2016; BISHOP, 2006).

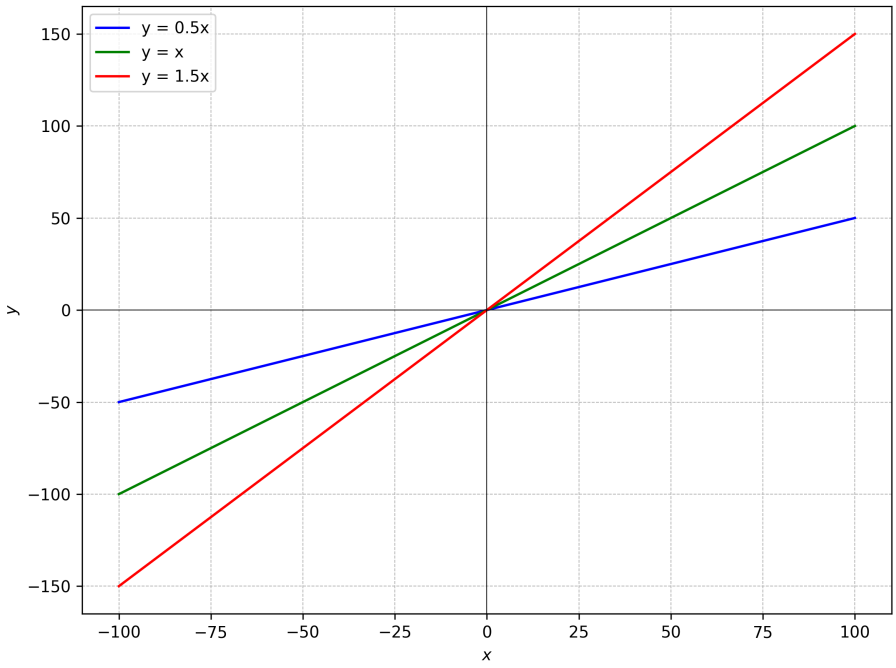


Figura 3 – Saídas de neurônios para diferentes valores de w, com *bias* nulo. Figura elaborada pelo autor.

Na ausência de funções de ativação, por propriedades de linearidade, a rede neural poderia ser vista apenas como uma combinação linear de suas entradas, independentemente do número

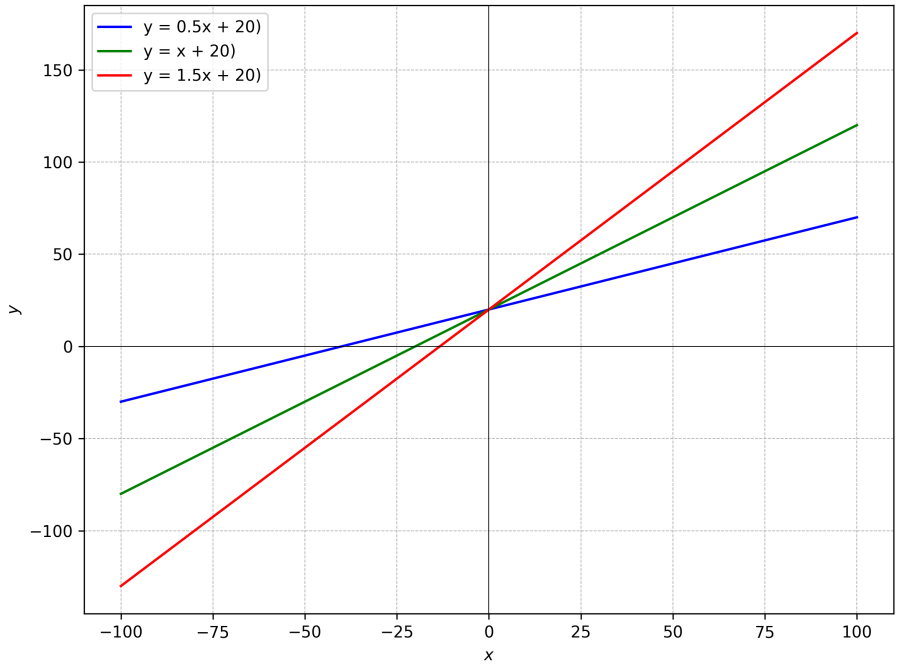


Figura 4 – Saídas de neurônios para diferentes valores de w, com *bias* igual a 20. Figura elaborada pelo autor.

de camadas. As funções de ativação introduzem não linearidade, permitindo que a rede aprenda e modele relações complexas (HORNIK; STINCHCOMBE; WHITE, 1989).

Entre as funções de ativação mais comuns, destacam-se a função sigmoide, a tangente hiperbólica (*tanh*) e a ReLU (*Rectified Linear Unit*).

A função sigmoide é uma das mais populares em tarefas de classificação binária e regressão logística. Sua forma é determinada por:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

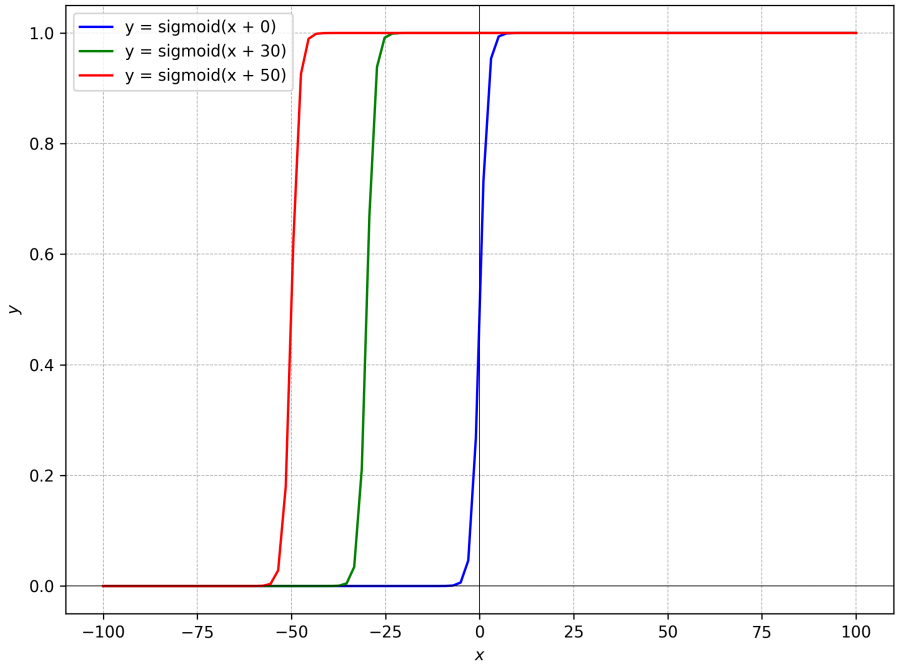


Figura 5 – Aplicação da função sigmoid para diferentes valores de *bias*. Figura elaborada pelo autor.

Entretanto, identificou-se que essa função é propensa ao problema do *vanishing gradient*, onde gradientes muito pequenos nas extremidades (próximas de 0 e 1) dificultam o treinamento de redes profundas (HOCHREITER; SCHMIDHUBER, 1997).

Outra função amplamente utilizada no passado, e que também compartilha do problema do *vanishing gradient*, é a função tangente hiperbólica (*tanh*). Sua forma é dada por:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.3}$$

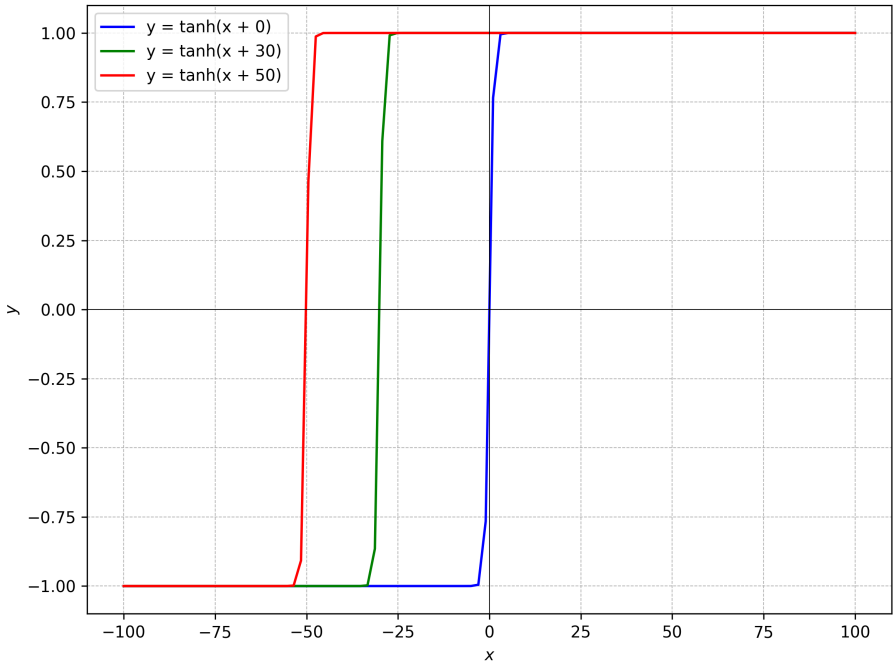


Figura 6 – Aplicação da função *tanh* para diferentes valores de *bias*. Figura elaborada pelo autor.

Assim como a sigmoide, a *tanh* apresenta um formato em S, mas mapeia os valores de entrada para um intervalo entre -1 e 1. Esse mapeamento centrado em torno de zero pode facilitar o treinamento em comparação com a sigmoide (LECUN et al., 1998).

No contexto das redes neurais profundas modernas, a ReLU (*Rectified Linear Unit*) tem-se tornado a função de ativação padrão na maioria dos casos:

$$\text{ReLU}(x) = \max(0, x) \tag{2.4}$$

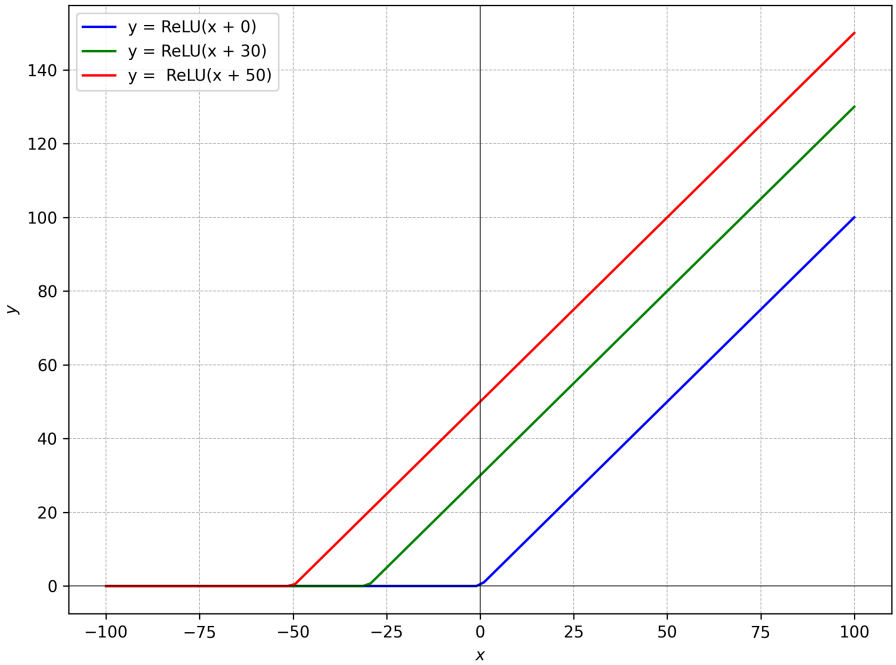


Figura 7 – Aplicação da função ReLU para diferentes valores de *bias*. Figura elaborada pelo autor.

Essa função retorna 0 para valores de entrada negativos e o próprio valor para valores positivos. A ReLU é computacionalmente eficiente e ajuda a mitigar o problema do *vanishing gradient*, permitindo que redes muito profundas sejam treinadas de forma mais eficaz (NAIR; HINTON, 2010).

Em arquiteturas ainda mais profundas, é comum o uso de técnicas adicionais para estabilizar o treinamento, como a normalização por lote (*batch normalization*) (IOFFE; SZEGEDY, 2015), que reduz a chamada *covariate shift* interna, e a técnica de *dropout* (SRIVASTAVA et al., 2014), que zera aleatoriamente as saídas de certos neurônios durante o treinamento, atuando como forma de regularização para evitar sobreajuste (*overfitting*).

Um MLP é composto por várias camadas de neurônios: a primeira conhecida como camada de entrada, que recebe diretamente os vetores de características do conjunto de dados; a última como camada de saída, a qual produz a estimativa final para a tarefa específica (por exemplo, probabilidade de classe ou valor predito); e as demais como camadas ocultas, responsáveis pela extração de características em diferentes níveis de abstração.

A camada de entrada não aplica nenhuma transformação aos dados, apenas transmite a informação para a primeira camada oculta. Ela pode ser descrita como

$$\mathbf{x} = [x_1, x_2, \ldots, x_n] \tag{2.5}$$

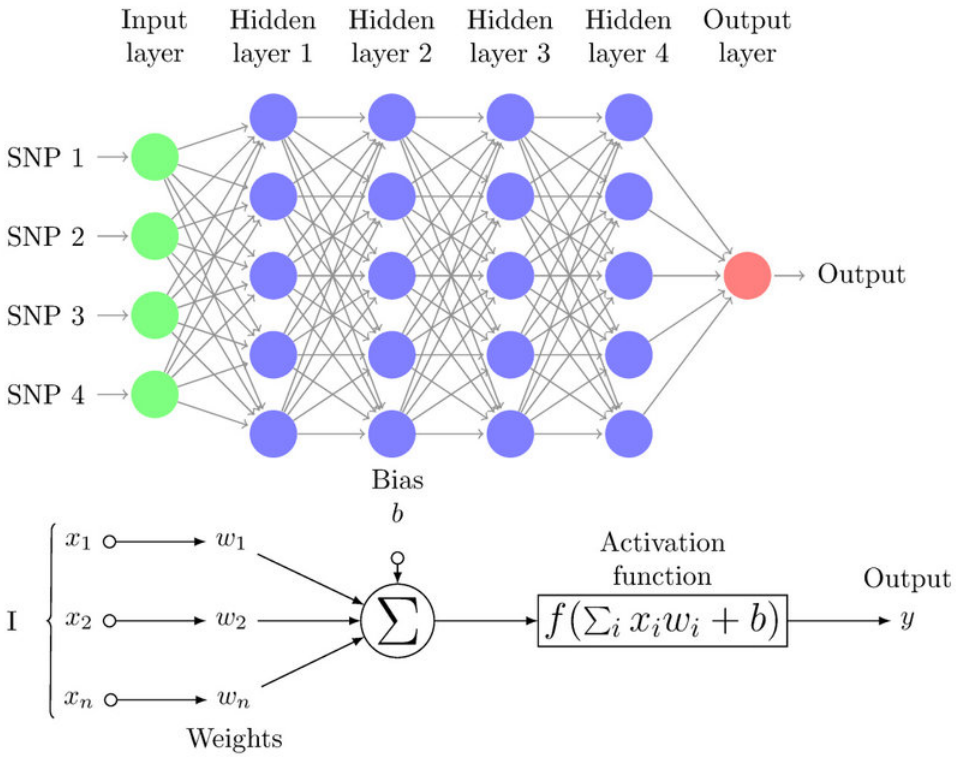


Figura 8 – Representação de uma MLP com 4 camadas ocultas. (ALRASHIDE et al., 2023)

onde cada x_i no vetor de entrada $\mathbf{x}$ representa uma característica ou atributo específico dos dados de entrada. Em tarefas de visão computacional, por exemplo, cada x_i poderia representar a intensidade de um pixel.

As camadas ocultas são camadas intermediárias da rede; nelas, cada neurônio recebe entradas ponderadas dos neurônios da camada anterior e aplica uma função de ativação. Isso permite que a rede aprenda a detectar padrões complexos e características hierárquicas nos dados. Múltiplas camadas ocultas caracterizam as chamadas redes profundas (*deep neural networks*).

Cada camada oculta é composta por vários neurônios, e a estrutura típica de uma camada oculta pode ser descrita como uma generalização da Equação (2.1):

$$h_j^{(l)} = \phi^{(l-1)}\left(\sum_{i=1}^{n^{(l-1)}} W_{ji}^{(l)}\, h_i^{(l-1)} + b_j^{(l)}\right) \tag{2.6}$$

onde o índice $j \in \{1, 2, \ldots, n^{(l)}\}$ identifica um neurônio específico da camada $l \in \{0, 1, \ldots, L\}$, e $h_j^{(l)} \in \mathbb{R}$ é a saída do mesmo. O termo $i \in \{1, 2, \ldots, n^{(l-1)}\}$ identifica um neurônio específico na camada anterior $(l-1)$. Desta maneira, $h_i^{(l-1)} \in \mathbb{R}$ representa a saída do neurônio i na camada anterior. Consequentemente, $W_{ji}^{(l)} \in \mathbb{R}$ pondera a saída desse neurônio anterior

para conectar ao neurônio atual. Por fim, $b_j^{(l)} \in \mathbb{R}$ identifica o viés do neurônio j na camada l. A função $\phi^{(l-1)} : \mathbb{R} \to \mathbb{R}$ é a função de ativação utilizada na camada anterior.

A camada de saída é a última camada de uma rede neural e sua função principal é transformar as ativações das camadas ocultas em uma forma apropriada para a tarefa específica (classificação, regressão etc.). Dependendo do problema, a camada de saída terá diferentes características e funções de ativação.

A formulação da camada de saída segue o padrão da Equação (2.6), porém com algumas especificidades:

$$h_k^{(L)} = \psi\left(\sum_{i=1}^{n^{(L-1)}} W_{ki}^{(L)}\, h_i^{(L-1)} + b_k^{(L)}\right), \tag{2.7}$$

onde $l = L$, por se tratar da última camada, e tanto o domínio de k quanto a função ψ são definidos de acordo com a tarefa.

- **Classificação binária**: Geralmente se utiliza apenas um neurônio na camada de saída, $k = \{1\}$, e a função de ativação mais comum é a sigmoide, que mapeia a saída para $(0, 1)$, interpretada como probabilidade de classe positiva.

- **Classificação multiclasse**: Há um neurônio de saída para cada classe. Se há K classes, então $k \in \{1, 2, \ldots, K\}$. A função de ativação *softmax* é comumente usada para normalizar as saídas em um vetor de probabilidades que somam 1, informando a confiança do modelo em classificar a entrada para cada uma das classes:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.8}$$

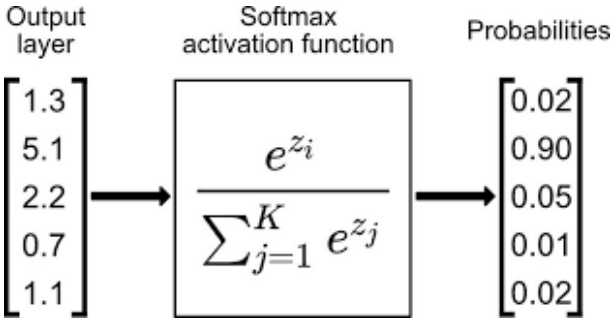


Figura 9 – Aplicação da função de ativação *softmax* (RADEčIć, 2024)

- **Regressão**: Para tarefas onde se busca prever um valor contínuo, a camada de saída costuma ter ativação linear (nenhum mapeamento não linear), permitindo que o va-

lor predito varie em toda a reta real. Caso haja restrições, outras funções podem ser empregadas, como *ReLU* ou *tanh*.

Abstraindo a dimensão do número de neurônios em cada camada l, podemos escrever o conjunto dos pesos como $\theta^l = \{W^l, b^l\}$, e o conjunto de todos os pesos de uma rede de L camadas como $\theta = \{\theta^1, \theta^2, \ldots, \theta^L\}$. Dessa forma, a saída de cada camada pode ser expressa como:

$$f^l = g(f^{(l-1)}, \theta^l) = \phi^l\Big(f^{(l-1)}W^l + b^l\Big), \tag{2.9}$$

onde para a primeira camada, $f^0 = \mathbf{x}$. O resultado final da rede $(\hat{y})$ pode ser visto como a composição de todas as transformações das L camadas: $\hat{y} = f^L \circ f^{L-1} \circ \ldots \circ f^1(\mathbf{x})$ (HAMMER; VILLMANN, 2003).

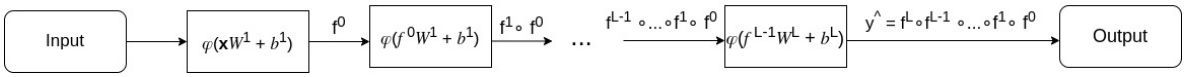


Figura 10 – Ilustração de uma rede neural como uma composição de funções. Figura elaborada pelo autor.

Processo de Aprendizado

Para compreender o processo de aprendizado, é fundamental entender também o papel da base de dados. Definimos um *dataset* $\mathcal{D}$ com N exemplos como:

$$\mathcal{D} = \{(x_i, y_i) \mid i = 1, 2, \ldots, N\},$$

onde

- $x_i = [x_{i1}, x_{i2}, \ldots, x_{id}]$ é o vetor de características do exemplo i, contendo d elementos, também chamado de *input*;
- y_i é o rótulo ou valor associado ao exemplo i, também conhecido como *groundtruth* ou *target*.

Tradicionalmente, a base de dados é dividida em 3 subconjuntos:

1. **Base de treinamento** ($\mathcal{D}_{\text{train}}$): usada para ajustar os pesos do modelo.
2. **Base de validação** ($\mathcal{D}_{\text{val}}$): usada para avaliar o desempenho do modelo durante o treino e ajustar hiperparâmetros.
3. **Base de testes** ($\mathcal{D}_{\text{test}}$): usada para avaliar o desempenho final, após o término do treinamento. É imprescindível que esse conjunto permaneça inédito para o modelo.

Para um certo x_i, o modelo pode fornecer uma estimativa $\hat{y}$, processando x_i camada a camada até a camada de saída (*feedforward*). A composição $f_L \circ f_{L-1} \circ \cdots \circ f_1(x)$ reflete esse fluxo de dados através das camadas, possibilitando que o modelo extraia progressivamente características mais complexas.

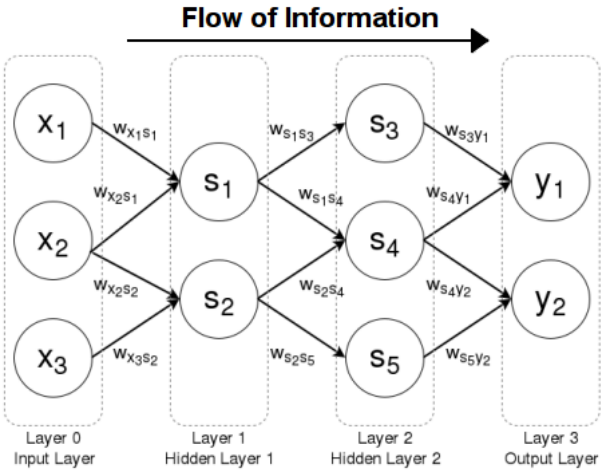


Figura 11 – Processo de *feedforward*. (BRILLIANT.ORG, 2024)

O método de aprendizado consiste em buscar pelo conjunto de pesos θ que minimize uma função de perda $\mathcal{L}(\theta, \mathcal{D})$, a qual estima a discrepância entre os *targets* e as estimativas da rede. Uma das funções de erro mais populares para problemas de regressão é o Erro Quadrático Médio (*Mean Squared Error* – MSE):

$$\mathcal{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2. \tag{2.10}$$

Para tarefas de classificação, a *Cross-Entropy* (ou *Log Loss*) costuma ser a escolha predominante, principalmente quando há uma interpretação probabilística da saída (ex.: uso de *softmax*). Pesquisas recentes mostram que a escolha da função de perda (*loss function*) impacta diretamente o desempenho e a capacidade de generalização do modelo, sendo considerada um hiperparâmetro crucial (DRäGER; DUNKELAU, 2022).

Uma vez definida a função de perda, o objetivo do treinamento é minimizar $\mathcal{L}$:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}). \tag{2.11}$$

A busca pela solução é tratada como um problema de otimização. O método mais comum é o *Stochastic Gradient Descent* (SGD) (ROBBINS; MONRO, 1951), que faz atualizações iterativas nos pesos. Em cada intervalo de tempo (época), o algoritmo atualiza o conjunto de pesos da seguinte forma:

$$\theta \leftarrow \theta - \eta \, \nabla_\theta \, \mathcal{L}(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}), \tag{2.12}$$

onde η é a **taxa de aprendizado** (*learning rate*), um hiperparâmetro que controla o tamanho do passo dado na direção do gradiente. Valores muito pequenos podem tornar o treinamento

lento, enquanto valores muito grandes podem levar à divergência. Na prática, estratégias como *mini-batch gradient descent* são frequentemente utilizadas para um melhor compromisso entre estabilidade e eficiência computacional.

Adicionalmente, métodos de ajuste adaptativo, como o *Adam* (KINGMA; BA, 2014), têm sido amplamente empregados por adaptarem a taxa de aprendizado de forma dinâmica, levando a uma convergência mais estável. Também se destacam técnicas como *momentum* (RUMELHART; HINTON; WILLIAMS, 1986) e *RMSProp* (HINTON, 2012), que ajudam a suavizar oscilações e acelerar a convergência no espaço de parâmetros.

A taxa de aprendizado está diretamente relacionada ao comportamento do gradiente e desempenha um papel crucial no processo de treinamento de redes neurais. Taxas de aprendizado mais altas podem acelerar o treinamento ao permitir atualizações maiores nos pesos, ajudando a explorar o espaço de solução de forma mais eficiente, especialmente em estágios iniciais do treinamento ou em cenários onde os gradientes não são excessivamente pequenos (BENGIO, 2012). No entanto, valores altos demais podem levar a oscilações em torno do mínimo ou mesmo à divergência do modelo, especialmente em casos de gradientes instáveis (PASCANU; MIKOLOV; BENGIO, 2013).

Por outro lado, taxas de aprendizado mais baixas promovem estabilidade no treinamento, permitindo que o modelo refine os pesos de maneira mais precisa nas proximidades de mínimos locais ou globais. Essa abordagem é particularmente benéfica em estágios avançados do treinamento ou em problemas com *exploding gradients*, onde valores menores ajudam a evitar atualizações excessivas e garantem convergência (KINGMA; BA, 2014). Adicionalmente, estratégias adaptativas como *Adam* ajustam dinamicamente a taxa de aprendizado, equilibrando esses benefícios em diferentes estágios do treinamento (KINGMA; BA, 2014). Estratégias de agendamento da taxa de aprendizado (*learning rate scheduling*), como o decaimento exponencial (SENIOR et al., 2013) ou o agendamento cíclico (SMITH, 2017), são comuns para mitigar esses problemas e refinar a convergência.

O operador ∇ simboliza o gradiente, um vetor contendo as derivadas parciais da função de perda $\mathcal{L}$ em relação a cada parâmetro. Ele representa tanto a direção de maior aumento da perda quanto a sensibilidade de cada peso à perda. Para diminuir $\mathcal{L}$, as atualizações são feitas na direção oposta ao gradiente:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}.$$

Especificamente, para um peso $\theta_{ji}^{(l)}$ na camada l, a atualização segue

$$\theta_{ji}^{(l)} \leftarrow \theta_{ji}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial \theta_{ji}^{(l)}}.$$

Esse ajuste é realizado por meio do processo de *backpropagation*, no qual o erro é propagado das camadas finais até as iniciais, permitindo o cálculo das derivadas parciais em todas as camadas.

Épocas e batches

O treinamento de uma rede neural ocorre iterativamente, normalmente organizado em **épocas** (*epochs*) e subdividido em ***batches*** de dados:

- **Época**: uma passagem completa por todo o conjunto de dados de treinamento. Em cada época, as amostras passam pelo modelo (*feedforward*) e têm seus pesos atualizados (*backpropagation*). Um número de épocas muito elevado pode levar ao sobreajuste, pois o modelo aprende detalhes específicos do conjunto de treinamento.

- ***batches***: Os dados normalmente não são processados todos de uma só vez, mas divididos em pequenos subconjuntos chamados *batches*. Cada *batch* contém um número fixo de amostras (*batch size*), e a atualização dos pesos é realizada com base nas informações extraídas desse subconjunto. Essa abordagem combina as vantagens do *batch gradient descent*, que utiliza todo o conjunto de dados para obter atualizações estáveis, com a agilidade do *stochastic gradient descent* puro, que atualiza os pesos após cada exemplo. Além disso, o uso de *batches* reduz a variância na estimativa do gradiente, resultando em uma convergência mais eficiente (LECUN; RANZATO, 2012). Também permite uma utilização mais eficaz de hardware moderno, como GPUs, otimizando o processamento paralelo de múltiplos exemplos (GOODFELLOW; BENGIO; COURVILLE, 2016).

Durante o treinamento, diversas técnicas de regularização podem ser aplicadas para evitar que o modelo se ajuste excessivamente aos dados de treinamento (*overfitting*), comprometendo sua capacidade de generalização. Métodos como a penalização L2 (NG, 2004), *dropout* (SRIVASTAVA et al., 2014) e *early stopping* (PRECHELT, 1998) são amplamente utilizados para mitigar esse problema. Esses métodos ajudam a garantir que, ao final do processo de treina-

mento, a rede neural mantenha boa capacidade preditiva em dados não vistos, preservando a generalização na base de testes e cenários reais.

Além disso, o treinamento de redes neurais profundas pode enfrentar desafios relacionados à escala dos gradientes e à escolha adequada da taxa de aprendizado. Dois problemas comuns nesse contexto são: (i) a instabilidade numérica ao utilizar precisão reduzida (*mixed precision training*), o que pode levar a valores subnormais ou *underflow*; e (ii) a necessidade de ajustes dinâmicos da taxa de aprendizado ao longo do treinamento para evitar estagnação ou oscilações excessivas.

Ao utilizar precisão mista (FP16), operações de ponto flutuante podem resultar em gradientes muito pequenos para serem representados com precisão, levando a perdas de informação. Para mitigar esse problema, o *GradScaler* ajusta dinamicamente a escala dos gradientes antes da retropropagação, garantindo que os valores permaneçam dentro de uma faixa representável. Esse procedimento melhora a estabilidade numérica e permite o uso eficiente de FP16 sem comprometer a precisão do modelo. (MICIKEVICIUS et al., 2017)

Estratégias de agendamento da taxa de aprendizado (*learning rate scheduling*) são frequentemente empregadas para refinar o treinamento ao longo das épocas. O método *StepLR* reduz a taxa de aprendizado de maneira programada, multiplicando-a por um fator fixo após um número pré-definido de épocas. Esse ajuste permite um treinamento mais controlado, prevenindo oscilações e melhorando a convergência do modelo. (SMITH, 2015)

2.2.2 Redes Neurais Convolucionais

As Redes Neurais Convolucionais (*Convolutional Neural Networks* – CNNs) foram introduzidas no final dos anos 1980 (LECUN et al., 1989; LECUN et al., 1998), mas sua adoção em larga escala se consolidou apenas na segunda década dos anos 2000, especialmente após o desempenho revolucionário demonstrado em competições de visão computacional como o ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (KRIZHEVSKY; SUTSKEVER; HINTON, 2012b; RUSSAKOVSKY et al., 2015). Nesse período, embora já houvesse evidências do potencial das CNNs, as limitações de *hardware* restringiam suas aplicações. O surgimento de arquiteturas de GPU/TPU (JOUPPI et al., 2017) e a disponibilidade de grandes bases de dados, como o ImageNet (DENG et al., 2009), foram fundamentais para que se alcançassem resultados notáveis em tarefas de classificação e detecção de objetos.

O sucesso das CNNs despertou intenso interesse tanto na academia quanto na indústria,

impulsionando investimentos em pesquisa e o desenvolvimento de aplicações comerciais. A crescente demanda por processamento de imagens e vídeos acelerou a evolução das técnicas de aprendizagem profunda. (LECUN; BENGIO; HINTON, 2015)

Diferenças em relação às Redes Neurais Artificiais (ANNs)

Enquanto as Redes Neurais Artificiais tradicionais, frequentemente implementadas como Redes Multicamadas Perceptrons (MLPs), utilizam camadas totalmente conectadas, nas quais cada unidade de processamento em uma camada está conectada a todas as unidades da camada subsequente (MCCULLOCH; PITTS, 1943; ROSENBLATT, 1958), as Redes Neurais Convolucionais (CNNs) introduzem conceitos fundamentais que transformaram a eficiência e a eficácia no processamento de dados visuais (LECUN et al., 1998). Diferentemente das MLPs, onde a conectividade total pode levar a um número excessivo de parâmetros em entradas de alta dimensionalidade, como imagens, as CNNs utilizam conexões locais e o compartilhamento de pesos, estratégias que otimizam o aprendizado e a generalização.

As conexões locais nas CNNs restringem a interação de cada unidade de processamento em uma camada convolucional a uma região limitada da entrada, conhecida como *campo receptivo*. Essa abordagem permite que a rede aprenda padrões locais, como bordas, texturas ou formas, capturando características relevantes sem depender diretamente de toda a informação global da entrada. Por exemplo, na detecção de bordas, é suficiente analisar pequenas regiões da imagem, reduzindo a complexidade computacional sem comprometer a extração de informações cruciais.

Adicionalmente, as CNNs implementam o compartilhamento de pesos, no qual um mesmo conjunto de parâmetros, representado pelo *kernel* ou filtro, é aplicado de forma sistemática a diferentes posições da entrada. Essa técnica não apenas reduz significativamente o número de parâmetros treináveis, mas também assegura que os padrões detectados, como bordas ou texturas específicas, sejam reconhecidos de maneira consistente em toda a extensão da imagem (LECUN et al., 1998; KRIZHEVSKY; SUTSKEVER; HINTON, 2012b). Em contrapartida, nas MLPs, cada conexão possui seu próprio peso, resultando em modelos altamente parametrizados que podem se tornar ineficientes ao lidar com entradas de grande dimensionalidade (GOODFELLOW; BENGIO; COURVILLE, 2016).

Essas inovações tornaram as CNNs particularmente eficazes na análise de dados estruturados, como imagens e vídeos, estabelecendo-as como uma das arquiteturas mais bem-sucedidas

para tarefas de visão computacional e aprendizado profundo.

O Operador de Convolução

O operador de convolução é o núcleo que permite às CNNs capturar padrões locais de forma hierárquica. Em Processamento Digital de Sinais (DSP), a convolução combina duas funções para enfatizar características relevantes. A operação contínua é definida por:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau, \tag{2.13}$$

e, para sinais discretos, a convolução é definida como uma soma. No contexto de imagens, a convolução bidimensional aplica o kernel K a um mapa de características I:

$$S(i,j) = (I * K)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i+m,\ j+n)\, K(m,n), \tag{2.14}$$

onde:

- $S(i,j)$ é o valor resultante na posição (i,j);
- M e N são as dimensões do kernel K;
- A expressão $I(i+m, j+n)$ representa o *campo receptivo* da entrada sobre o qual o kernel atua.

A Figura 12 ilustra esse processo, demonstrando como o kernel se desloca sobre a imagem e realiza operações de multiplicação e soma para extrair características locais.

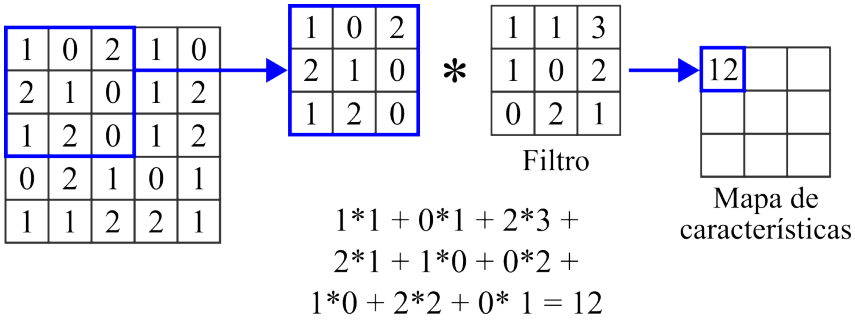


Figura 12 – Exemplo da operação de convolução. O kernel se move sobre a imagem, combinando valores locais para formar o mapa de características. (ALCANTARA, 2024)

Detalhamento sobre Stride, Padding e Pooling

A eficiência e a eficácia da extração de características em CNNs dependem também de hiperparâmetros como *stride*, *padding* e *pooling*. Esses parâmetros controlam a forma como o kernel interage com a entrada e como as informações são consolidadas ao longo da rede.

Stride

O *stride* especifica o número de *pixels* que o kernel salta ao se deslocar pela imagem. Um *stride* maior implica que o kernel analise menos posições, reduzindo assim a dimensão do mapa de características e o custo computacional. No entanto, isso pode resultar em perda de detalhes importantes. Por sua vez, um *stride* menor garante uma análise mais densa da imagem, preservando mais detalhes espaciais. A Figura 13 demonstra a diferença entre um *stride* de 1 e um *stride* de 2.

(a) Passo = 1 (b) Passo = 2

Figura 13 – Comparação entre diferentes tamanhos de *stride*: análise densa versus redução de dimensão. (ALCANTARA, 2024)

Padding

O *padding* consiste na adição de pixels, geralmente zeros (*zero-padding*), às bordas da entrada, permitindo que o kernel seja aplicado também nas regiões periféricas da imagem. Essa técnica é crucial para preservar a dimensão espacial da imagem na saída, garantindo que o tamanho das representações internas não seja excessivamente reduzido após as operações de convolução (LECUN et al., 1998). Além disso, o *padding* aumenta o campo de visão da rede, possibilitando que a rede "veja"um contexto maior nas bordas e, consequentemente, capture características relevantes que podem estar localizadas nas regiões extremas da imagem (GOODFELLOW; BENGIO; COURVILLE, 2016). Aplicações típicas que se beneficiam do *padding* incluem tarefas em que a posição espacial da informação é crítica, como segmentação de imagens e

detecção de objetos, onde o posicionamento preciso das bordas pode ser determinante para a precisão das predições (LECUN; BENGIO; HINTON, 2015).

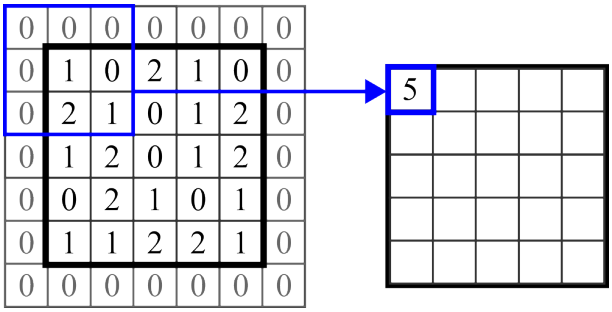


Figura 14 – Ilustração do efeito da aplicação de preenchimento. (ALCANTARA, 2024)

Pooling

As camadas de *pooling* desempenham um papel essencial na redução da dimensionalidade dos mapas de características, consolidando informações e tornando a rede mais robusta a pequenas variações e deslocamentos na entrada. Essa redução não apenas diminui a complexidade computacional, mas também ajuda a evitar o overfitting, proporcionando uma certa invariância a transformações locais (LECUN et al., 1998). Existem diferentes tipos de *pooling*, entre os quais se destacam o *Max Pooling*, o *Average Pooling* e o *Global Average Pooling*. O *Max Pooling* seleciona o valor máximo dentro de uma janela, por exemplo, 2×2, enfatizando a presença das características mais fortes na região analisada. Por outro lado, o *Average Pooling* calcula a média dos valores dentro da janela, proporcionando uma visão mais suavizada e geral da região. O *Global Average Pooling* reduz cada mapa de características a um único valor, sendo comumente utilizado para conectar a parte de extração de características à camada final de classificação (GOODFELLOW; BENGIO; COURVILLE, 2016). Essas técnicas são fundamentais para consolidar informações relevantes e garantir que a rede aprenda representações robustas das entradas, facilitando a generalização para dados não vistos anteriormente. A Figura 15 exemplifica o processo de *Max-Pooling*, ilustrando como o valor máximo é selecionado dentro de uma janela específica.

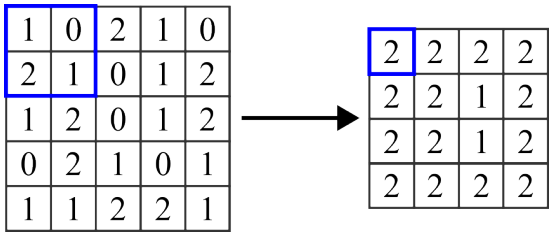


Figura 15 – Ilustração da operação de *Max-Pooling*: a janela 2×2 seleciona o valor máximo em cada região. (ALCANTARA, 2024)

Transformação de uma Camada Convolucional

Em redes neurais convencionais (ANNs), a transformação de uma camada ocorre através de um produto matricial seguido de uma função de ativação. Em CNNs, essa transformação é realizada pela operação de convolução combinada com o acréscimo de um termo de *bias*, aplicação de uma função de ativação (como ReLU) e, frequentemente, uma operação de *pooling* para redução da dimensionalidade. Formalmente, para o mapa de características $h_j^{(l)}$ na camada l, pode-se escrever:

$$h_j^{(l)} = \text{Pooling}\Big(\phi\Big((W_j^{(l)} * h^{(l-1)}) + b_j^{(l)}\Big)\Big), \tag{2.15}$$

onde:

- $W_j^{(l)}$ é o filtro associado ao j-ésimo canal de saída;
- $b_j^{(l)}$ é o termo de *bias*;
- $\phi(\cdot)$ representa a função de ativação, por exemplo, ReLU;
- Pooling$(\cdot)$ é a operação de *pooling*, quando aplicada.

Estrutura Geral e Definição Matemática do Backbone

Extrator de Características

Após a aplicação sequencial de diversas camadas convolucionais – intercaladas com funções de ativação e operações de pooling – a rede gera um conjunto de mapas de características que representam informações em diferentes níveis de abstração. Esse conjunto de operações forma o núcleo ***extrator de características*** da CNN.

Conceitualmente, o *extrator de características* desempenha duas funções principais. Primeiramente, nas primeiras camadas, a rede captura informações básicas, como bordas, texturas e formas. À medida que a profundidade aumenta, essas informações são combinadas para formar representações mais complexas e globais do conteúdo da imagem. Em segundo lugar, por meio de operações de pooling e convoluções sucessivas, o *extrator de características* reduz a dimensão espacial dos dados, preservando uma representação robusta e invariável a pequenas variações (LECUN; BENGIO; HINTON, 2015; GOODFELLOW; BENGIO; COURVILLE, 2016).

Para formalizar o *extrator de características* matematicamente, considere:

- x como a entrada (por exemplo, uma imagem);
- $f^{(i)}(\cdot)$ como a transformação realizada pela i-ésima camada do *extrator de características* – que pode incluir convolução, função de ativação e pooling;
- $\theta^{(i)}$ os parâmetros (filtros, biases, etc.) associados à i-ésima camada.

Definimos, então, a saída de cada camada sequencialmente:

$$h^{(0)} = x, \tag{2.16}$$

e, para $i = 1, 2, \ldots, k$:

$$h^{(i)} = f^{(i)}\left(h^{(i-1)}; \theta^{(i)}\right). \tag{2.17}$$

A composição completa dessas transformações define o *extrator de características* da rede:

$$\mathcal{B}(x) = h^{(k)} = \left(f^{(k)} \circ f^{(k-1)} \circ \cdots \circ f^{(1)}\right)(x). \tag{2.18}$$

Nesta formulação, $h^{(k)}$ representa a extração final de características, uma representação rica e hierárquica do dado de entrada, que pode ser utilizada por camadas adicionais (por exemplo, totalmente conectadas e softmax) para realizar a predição final:

$$\hat{y} = \Upsilon\left(h^{(k)}\right). \tag{2.19}$$

Onde a função Υ pode ser interpretada como o *head* da rede. Em outras palavras, é a parte final do modelo que recebe como entrada as representações aprendidas pelo *extrator de características* (ou *backbone*) e produz a predição $\hat{y}$. Geralmente, esse *head* consiste em uma ou mais camadas totalmente conectadas (FC) seguidas de uma função de ativação apropriada (por exemplo, *softmax* para classificação ou uma função linear para regressão). Dessa forma,

Υ mapeia a saída $h^{(k)}$ do *extrator de características* para o espaço de predição, gerando o resultado final $\hat{y}$.

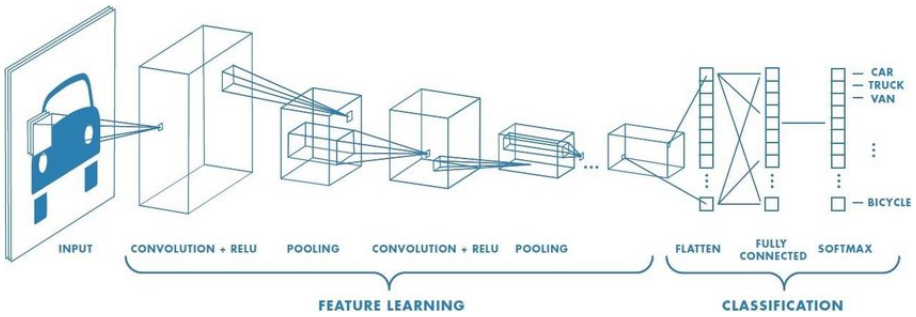


Figura 16 – Diagrama ilustrativo do pipeline de extração de características em uma CNN, onde as camadas iniciais extraem informações locais e as mais profundas combinam essas informações para formar um *embedding* discriminativo. (SAHA, 2018)

A Figura 16 ilustra esse fluxo, mostrando a entrada passando pelo *extrator de características* – com suas camadas convolucionais, funções de ativação e pooling – até a transição para as camadas finais responsáveis pela predição.

2.2.3 Extratores de Características

Extratores de características referem-se a técnicas ou módulos dentro de redes neurais capazes de mapear dados de entrada para representações de maior nível de abstração, geralmente na forma de vetores ou tensores compactos que preservam as propriedades essenciais da informação original. Em modelos de visão computacional, por exemplo, redes convolucionais (CNNs) treinam filtros que identificam, progressivamente, padrões locais — como bordas, texturas e formas — e, em camadas mais profundas, combinam essas informações para detectar partes de objetos ou até objetos completos. Dessa forma, as camadas finais de uma CNN bem treinada funcionam como um extrator de características robusto, capaz de fornecer *embeddings* representativos que podem ser reutilizados em diferentes tarefas.

Conceitualmente, o *extrator de características* desempenha um papel fundamental na capacidade das CNNs de generalizar e adaptar-se a diversas aplicações. Dentro de arquiteturas mais complexas, especialmente em modelos de detecção de objetos, o termo ***backbone*** é frequentemente utilizado para designar a parte central da rede responsável pela extração de características. O *backbone* atua como o *extrator de características*, processando a imagem de entrada através de múltiplas camadas convolucionais e de *pooling* para gerar representações

hierárquicas que capturam informações em diferentes níveis de abstração (LECUN; BENGIO; HINTON, 2015; HE et al., 2016).

Em modelos de detecção de objetos, como Faster R-CNN (REN et al., 2015), SSD (LIU et al., 2016) e YOLO (REDMON et al., 2016), o *backbone* é tipicamente uma arquitetura de rede neural pré-treinada, como ResNet (HE et al., 2016), VGG (SIMONYAN; ZISSERMAN, 2014) ou EfficientNet (TAN; LE, 2019). Essas arquiteturas são escolhidas por sua eficácia comprovada na extração de características robustas e discriminativas, essenciais para a identificação e localização precisa de objetos nas imagens. Ao utilizar um *backbone* pré-treinado, esses modelos de detecção aproveitam representações aprendidas em grandes conjuntos de dados, como o ImageNet, o que melhora significativamente o desempenho e reduz o tempo de treinamento para tarefas específicas (GOODFELLOW; BENGIO; COURVILLE, 2016).

Assim, enquanto os *backbones* funcionam como extratores de características, eles são especificamente integrados em arquiteturas de detecção para fornecer uma base sólida sobre a qual as camadas subsequentes operam, realizando a predição final dos objetos detectados. Em muitos casos, a rede utilizada como *backbone* é "**truncada**" em suas camadas finais de classificação, de modo a preservar apenas as convoluções e eventuais blocos intermediários responsáveis pela geração de mapas de ativação. Truncar uma rede neural consiste em interromper seu fluxo de processamento em um ponto específico — geralmente removendo as últimas camadas densas (totalmente conectadas) — e mantendo apenas a porção convolucional como extratora de recursos. Dessa forma, as representações produzidas podem ser facilmente reaproveitadas em módulos posteriores de detecção ou segmentação, sem a necessidade de treinar toda a arquitetura desde o início.

2.2.4 Principais Famílias de Extratores Baseados em CNN

Neste trabalho, destacam-se as seguintes arquiteturas: VGG, ResNet, EfficientNet-Lite e ShuffleNet.

2.2.4.1 Visual Geometry Group

A família VGG (SIMONYAN; ZISSERMAN, 2014) foi inicialmente proposta para classificação de imagens no ImageNet, mas sua influência rapidamente se espalhou para diversas aplicações em visão computacional. Uma das motivações centrais desse trabalho foi investigar sistemati-

camente o impacto do aumento da profundidade da rede e do uso de filtros convolucionais de tamanho reduzido (geralmente 3×3) no desempenho. A escolha de camadas convolucionais menores, repetidas várias vezes em sequência, permite capturar padrões mais refinados a cada estágio e facilita a análise dos efeitos da profundidade na capacidade de generalização do modelo.

Além do forte resultado obtido na competição ImageNet Large Scale Visual Recognition Challenge (ILSVRC)(RUSSAKOVSKY et al., 2015) em 2014, outro aspecto que contribuiu para a popularidade da VGG foi a adoção de uma arquitetura relativamente simples e sistemática, tornando-a uma espécie de "modelo de referência" para muitas pesquisas e aplicações em visão computacional. A VGG, com sua estrutura modular e bem definida, também oferece uma facilidade maior de adaptação, seja para tarefas de classificação, detecção de objetos ou segmentação semântica.

No entanto, mesmo com suas vantagens, a VGG é frequentemente apontada como relativamente custosa em termos de número de parâmetros e uso de memória, sobretudo quando comparada a arquiteturas mais recentes e eficientes. Ainda assim, graças à disponibilidade de pesos pré-treinados e à robustez na extração de características, a VGG permanece popular como *backbone* em diversos cenários: desde *pipelines* de transferência de estilo e recuperação de imagens até sistemas de detecção de anomalias e análise de dados médicos.

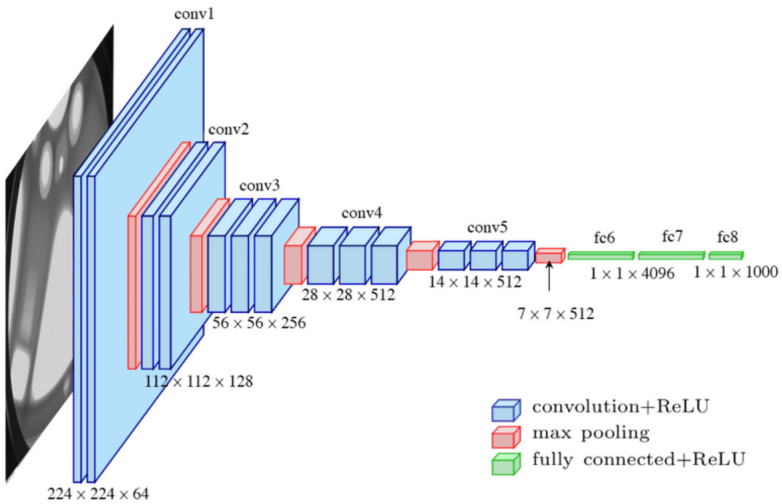


Figura 17 – Ilustração da arquitetura da VGG16, proposta por (SIMONYAN; ZISSERMAN, 2014)

Tabela 1 – Estrutura resumida da VGG16, destacando os blocos convolucionais e *pooling*

Bloco	Operador	Filtros	Stride/Pool	Saída Aproximada
Input	-	-	-	$224 \times 224 \times 3$
Conv Block 1	Conv 3×3 (2x), ReLU	64	Pool 2×2, Stride 2	$112 \times 112 \times 64$
Conv Block 2	Conv 3×3 (2x), ReLU	128	Pool 2×2, Stride 2	$56 \times 56 \times 128$
Conv Block 3	Conv 3×3 (3x), ReLU	256	Pool 2×2, Stride 2	$28 \times 28 \times 256$
Conv Block 4	Conv 3×3 (3x), ReLU	512	Pool 2×2, Stride 2	$14 \times 14 \times 512$
Conv Block 5	Conv 3×3 (3x), ReLU	512	Pool 2×2, Stride 2	$7 \times 7 \times 512$
FC Layers	FC (4096), ReLU	-	-	$1 \times 1 \times 1000$

Em aplicações específicas de detecção ou classificação, é comum "**truncar**" a VGG — removendo as camadas densas finais (FC, na Tabela 1) — para aproveitar somente os mapas de ativação produzidos pelas últimas camadas convolucionais. Essa estratégia permite que os descritores aprendidos (fortemente discriminativos) sejam fornecidos a módulos responsáveis pela detecção ou classificação de objetos. Exemplos notórios incluem abordagens como Fast R-CNN (GIRSHICK, 2015) e Faster R-CNN (REN et al., 2015), em que a rede VGG truncada provê uma base eficiente para os estágios de *region proposal* e classificação, reforçando a adaptabilidade e eficácia dessa arquitetura em diferentes cenários de visão computacional.

2.2.4.2 ResNet

A família ResNet (HE et al., 2016) representa um passo significativo no desenvolvimento de redes neurais convolucionais, pois introduziu o conceito de *blocos residuais* para contornar os problemas associados ao treinamento de arquiteturas muito profundas, sobretudo o desaparecimento de gradientes. Em redes anteriores, como a VGG (SIMONYAN; ZISSERMAN, 2014), já se observava que o aumento do número de camadas melhorava a capacidade de representação, mas o ganho na profundidade podia ser dificultado pelo fenômeno de *vanishing gradients*, que fazia com que atualizações de peso nas camadas iniciais fossem quase nulas depois de muitas iterações de retropropagação.

O diferencial das ResNets é a adoção de conexões de atalho (*skip connections*), que adicionam diretamente a saída de um conjunto de camadas convolucionais (representada por $F(x)$) à entrada original x, resultando em uma função de saída $y = F(x) + x$. Em outras palavras, o bloco residual foca em aprender a diferença entre a saída desejada e a entrada

$\Big(H(x) - x = F(x)\Big)$, em vez de tentar aproximar diretamente $H(x)$. Essa reformulação simplifica o processo de otimização, pois o gradiente encontra um caminho direto para retropropagar através das camadas iniciais, permitindo que se treine redes com profundidade significativamente maior (por exemplo, versões com 50, 101 ou 152 camadas).

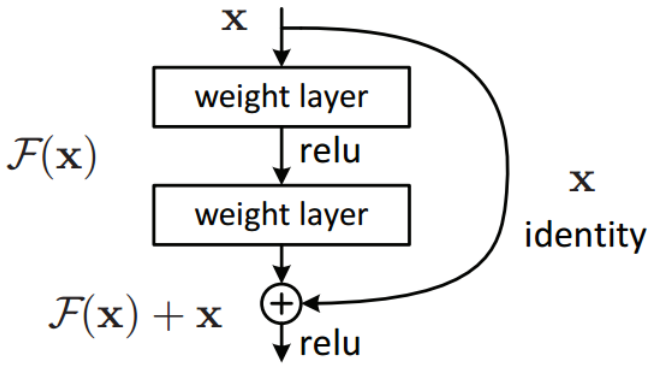


Figura 18 – Ilustração do bloco residual (HE et al., 2016)

Outro aspecto importante no desenho das ResNets, especialmente nas variantes mais profundas, é o uso de blocos denominados *bottleneck*. Esses blocos tipicamente empregam convoluções 1×1 para redução (e posterior restauração) da dimensionalidade antes e depois de convoluções 3×3, de modo a conter o número de parâmetros e facilitar o fluxo de informações. Dessa forma, redes como a ResNet101 ou ResNet152 conseguem equilibrar profundidade e custo computacional, tornando-se escaláveis para diferentes aplicações sem comprometer a capacidade de aprendizado.

A adoção desse paradigma de blocos residuais se mostrou eficaz na ILSVRC, onde as ResNets atingiram resultados superiores em tarefas de classificação. Não demorou para que o conceito fosse estendido a outras aplicações, como detecção e segmentação de objetos. Trabalhos como o Mask R-CNN (HE et al., 2017) evidenciam essa versatilidade ao utilizar versões truncadas das ResNets (isto é, sem as camadas densas finais) como *backbone* para extrair mapas de ativação sobre os quais módulos específicos de detecção e segmentação operam. De forma análoga, em abordagens de segmentação semântica, como o DeepLab (CHEN et al., 2017), as camadas finais são adaptadas para produzir mapas de características em resoluções maiores, preservando o cerne convolucional residual que lida com o aprendizado dos padrões espaciais.

O sucesso das ResNets também estimulou o surgimento de variações, como a ResNeXt (XIE et al., 2017) e as Wide ResNets (ZAGORUYKO; KOMODAKIS, 2016), que se baseiam na mesma ideia de blocos residuais, mas exploram outros fatores de arquitetura, como a "cardinalidade"

(múltiplos caminhos de convolução dentro do mesmo bloco) ou o aumento da largura das camadas. Todas essas adaptações mantêm o princípio fundamental de adicionar a entrada diretamente à saída do bloco convolucional.

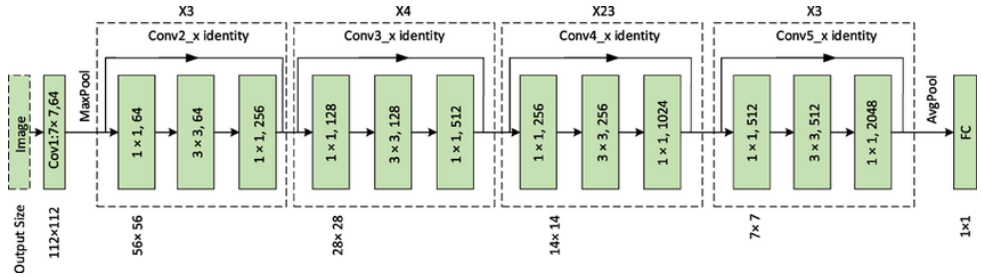


Figura 19 – Ilustração da ResNet101 (HE et al., 2016)

Assim como observado na VGG, a possibilidade de *truncar* a ResNet em suas camadas finais a torna uma excelente candidata para servir de *extrator de características*. O corpo residual, ao longo de suas múltiplas camadas, produz descritores robustos e hierárquicos, capturando desde bordas e texturas iniciais até composições mais complexas em níveis avançados da rede. Essas representações podem então alimentar módulos de classificação, detecção ou segmentação, permitindo que diversas aplicações em visão computacional se beneficiem do potencial de generalização das ResNets sem precisar iniciar um treinamento do zero em cada caso.

Tabela 3 – Resumo das camadas principais da ResNet101

Layer	**Blocos**	**Convoluções**	**Stride**	**Saída**
Conv Inicial	-	Conv 7×7, BN, ReLU	Stride 2	$112 \times 112 \times 64$
Layer1	3	$1 \times 1 \rightarrow 3 \times 3 \rightarrow 1 \times 1$	1	$56 \times 56 \times 256$
Layer2	4	$1 \times 1 \rightarrow 3 \times 3 \rightarrow 1 \times 1$	2 (1º bloco)	$28 \times 28 \times 512$
Layer3	23	$1 \times 1 \rightarrow 3 \times 3 \rightarrow 1 \times 1$	2 (1º bloco)	$14 \times 14 \times 1024$
Layer4	3	$1 \times 1 \rightarrow 3 \times 3 \rightarrow 1 \times 1$	2 (1º bloco)	$7 \times 7 \times 2048$
Pool + FC	-	Global Pool + FC (classes)	-	$1 \times 1 \times$ classes

2.2.4.3 EfficientNet

A família EfficientNet (TAN; LE, 2019) introduziu uma abordagem de escalonamento composto (*compound scaling*) para redes convolucionais, que unifica o ajuste de profundidade, largura e resolução de entrada em um único processo de otimização. Essa estratégia elimina a

necessidade de definir manualmente cada um desses fatores, pois estabelece um balanceamento sistemático que orienta o aumento (ou redução) homogêneo da complexidade do modelo de acordo com o recurso computacional disponível.

No cerne da arquitetura estão os *MBConv blocks*, também conhecidos como *Mobile Inverted Bottleneck Convolutions*, originados a partir dos princípios do MobileNet (HOWARD et al., 2017). Cada bloco MBConv inicia com uma camada de expansão (geralmente uma convolução 1×1) que aumenta o número de canais, seguida por uma convolução aprofundada (*depthwise convolution* 3×3 ou 5×5) e, por fim, uma camada de redução (nova convolução 1×1) que restabelece a dimensionalidade. Ao longo desse percurso, é comum que se adotem *skip connections* (atalhos) sempre que as dimensões de entrada e saída coincidirem, o que favorece a propagação de gradientes em redes mais profundas.

A variante EfficientNet-Lite busca otimizações adicionais para cenários com restrições de latência ou energia, substituindo componentes dispendiosos (como SE e Swish) por alternativas mais leves, a exemplo do uso de ReLU6. Na configuração EfficientNet-Lite0, mantém-se o mesmo processo de escalonamento composto, mas com hiperparâmetros ajustados para aparelhos móveis ou embarcados. Esse design conserva a eficiência dos blocos MBConv, mantendo a estrutura de expansão e contração de canais para balancear o custo computacional e a capacidade de representação, ao mesmo tempo que reduz operações que elevem o tempo de inferência em hardware modesto.

Assim como em redes como VGG ou ResNet, a EfficientNet-Lite pode ser "truncada" antes da camada final de classificação, preservando exclusivamente suas camadas convolucionais como *backbone* para extração de características. Esse recurso possibilita que tarefas como detecção de objetos ou segmentação se beneficiem dos mapas de ativação gerados pelos blocos MBConv, mantendo baixa a demanda de recursos. Nesse cenário, o potencial de generalização adquirido pelo treinamento em grandes bases de dados, somado à leveza do projeto, torna a EfficientNet-Lite atrativa para uma ampla gama de aplicações, em especial aquelas que exigem implementação em dispositivos de capacidade limitada.

Tabela 5 – Resumo das camadas da EfficientNet-Lite0

Estágio	Operador	Configuração	Stride	Saída
Input	Conv 3×3	$3 \rightarrow 32$, BN, ReLU6	2	$112 \times 112 \times 32$
MBConv1	MBConv	$32 \rightarrow 16$, 3×3	1	$112 \times 112 \times 16$
MBConv6	MBConv	$16 \rightarrow 24$, 3×3	2 (1º bloco)	$56 \times 56 \times 24$
MBConv6	MBConv	$24 \rightarrow 40$, 5×5	2 (1º bloco)	$28 \times 28 \times 40$
MBConv6	MBConv	$40 \rightarrow 80$, 3×3	2 (1º bloco)	$14 \times 14 \times 80$
MBConv6	MBConv	$80 \rightarrow 112$, 5×5	1	$14 \times 14 \times 112$
MBConv6	MBConv	$112 \rightarrow 192$, 5×5	2 (1º bloco)	$7 \times 7 \times 192$
Conv Final	Conv 1×1	$192 \rightarrow 1280$, BN, ReLU6	1	$7 \times 7 \times 1280$
Pool + FC	Global Pool + FC	FC (classes)	-	$1 \times 1 \times$ classes

2.3 REDES NEURAIS DINÂMICAS

Redes neurais dinâmicas representam uma classe de arquiteturas de aprendizado profundo projetadas para ajustar sua complexidade computacional durante a inferência, otimizando o uso de recursos computacionais de acordo com a complexidade de cada entrada. Diferentemente das redes estáticas, que mantêm uma estrutura e um número fixo de operações para todas as amostras, as redes dinâmicas ajustam seu comportamento de forma flexível, permitindo que diferentes amostras sejam processadas com distintos níveis de complexidade. Esse conceito se mostra especialmente útil em tarefas de visão computacional e em cenários com restrições de tempo real, como sistemas embarcados e veículos autônomos. (HAN et al., 2021)

A principal motivação por trás do uso de redes neurais dinâmicas é equilibrar o custo computacional e a precisão do modelo. Em tarefas práticas, como detecção de objetos e classificação de imagens, muitas amostras podem ser processadas corretamente com menos recursos, enquanto entradas mais complexas exigem análises mais profundas para garantir a qualidade da predição. Redes dinâmicas buscam explorar essa variação intrínseca de complexidade, otimizando a alocação de recursos.

Esses modelos podem ser organizados em três categorias principais: abordagens *sample-wise*, abordagens *spatial-wise* e abordagens *temporal-wise*, cada uma com estratégias distintas de adaptação (HAN et al., 2021).

Abordagem Sample-wise

A abordagem conhecida como *sample-wise* baseia-se na adaptação da arquitetura da rede ou de seus parâmetros para cada amostra individualmente. Em vez de aplicar o mesmo conjunto de operações a todas as entradas, a rede decide dinamicamente quais partes de sua estrutura serão ativadas ou desativadas, otimizando a complexidade computacional em função da complexidade da entrada processada.

Essa estratégia é frequentemente implementada por meio de mecanismos como saídas antecipadas (*early exits*), onde a rede interrompe a inferência quando uma saída intermediária atinge um nível de confiança suficiente. Modelos como *SkipNet* e *BlockDrop* também exploram a ideia de caminhos dinâmicos, nos quais blocos ou camadas específicas são ignorados ou processados seletivamente.

Essa abordagem apresenta vantagens claras em termos de eficiência computacional, uma vez que amostras mais simples são processadas mais rapidamente, reduzindo o tempo de inferência e o consumo de energia. Essa eficiência é particularmente útil em sistemas de inferência em tempo real, como veículos autônomos, onde decisões rápidas são cruciais para a segurança. No entanto, a introdução de saídas intermediárias e caminhos variáveis adiciona complexidade ao treinamento, exigindo mecanismos de calibração e otimização para assegurar que as decisões de interrupção antecipada não comprometam a precisão global. Modelos mal calibrados podem interromper a inferência prematuramente, resultando em predições menos confiáveis (HAN et al., 2021).

Abordagem Spatial-wise

A abordagem *spatial-wise* diferencia-se ao ajustar a alocação de recursos computacionais de acordo com diferentes regiões espaciais da entrada, normalmente uma imagem. Essa técnica explora o fato de que, em muitas tarefas de visão computacional, partes da imagem contêm mais informações relevantes do que outras. Por exemplo, em uma cena de trânsito, as áreas contendo veículos e pedestres são mais informativas para a detecção do que o fundo ou o céu, que podem ser processados com menor detalhamento.

Redes neurais com essa abordagem frequentemente utilizam mecanismos de atenção adaptativa, como *Spatial Transformer Networks* e redes com convoluções esparsas, que permitem concentrar o poder de processamento apenas nas regiões mais relevantes da entrada. Essa

alocação seletiva de recursos proporciona um ganho em eficiência computacional sem comprometer a precisão para tarefas onde a informação é espacialmente distribuída de forma desigual.

Essa abordagem é particularmente vantajosa em tarefas de segmentação e detecção de objetos, pois permite que a rede priorize áreas de interesse enquanto ignora informações irrelevantes. No entanto, sua complexidade de implementação é elevada, uma vez que requer a integração de módulos de controle espacial sofisticados e treinamento cuidadoso para garantir que nenhuma região crítica seja ignorada. Além disso, há o risco de que o modelo negligencie informações importantes em contextos onde a relevância das regiões não é claramente definível, como em imagens médicas complexas (HAN et al., 2021).

Abordagem Temporal-wise

A abordagem *temporal-wise* é voltada para o processamento de dados sequenciais ou temporais, como vídeos ou séries temporais. Em vez de processar todos os quadros de um vídeo ou todas as etapas de uma sequência de texto de forma uniforme, o modelo ajusta a profundidade ou a quantidade de operações aplicadas em cada passo temporal. Esse ajuste ocorre com base na complexidade ou importância de cada segmento da sequência.

Essa técnica é comumente utilizada em modelos de análise de vídeo, como aqueles que aplicam seleção de quadros-chave, onde apenas os quadros mais relevantes são processados com alta complexidade, enquanto quadros redundantes ou de menor importância são descartados ou processados com menor profundidade. Além disso, redes neurais recorrentes adaptativas ajustam a profundidade do ciclo de recorrência com base no conteúdo informativo da sequência.

O benefício central dessa abordagem é a redução substancial da complexidade temporal, o que é particularmente importante em tarefas de análise de vídeo em tempo real, como segurança por câmeras e diagnósticos médicos em vídeo. No entanto, essa redução pode ser acompanhada de perda de informações, especialmente se o modelo não for adequadamente treinado para identificar corretamente os segmentos mais relevantes da sequência. A implementação também é desafiadora, uma vez que a decisão de interromper o processamento ou reduzir o detalhamento temporal precisa ser aprendida de forma eficaz durante o treinamento (HAN et al., 2021).

AdaBoost e Early exit

O algoritmo *AdaBoost (Adaptive Boosting)*, proposto por Freund e Schapire (1997), realiza a combinação sequencial de modelos simples, cuja taxa de acerto é apenas um pouco melhor que a de uma escolha aleatória. O objetivo é formar um classificador duplamente mais robusto e preciso. Em cada iteração, o algoritmo atualiza uma distribuição de pesos sobre as amostras de treinamento, de modo que os exemplos incorretamente classificados recebem mais peso, enquanto os exemplos corretamente classificados têm seu peso reduzido. Dessa forma, o próximo classificador é treinado com foco maior nos exemplos que apresentaram maior dificuldade de classificação nas iterações anteriores. Essa lógica é formalizada pela seguinte regra de atualização:

$$D_{t+1}(i) = \frac{D_t(i) \cdot e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

em que $D_t(i)$ representa o peso da amostra i na iteração t, h_t é o classificador aprendido nesta etapa, α_t é o peso atribuído ao classificador com base em sua acurácia e Z_t é um fator de normalização que garante que D_{t+1} seja uma distribuição de probabilidade válida (FREUND; SCHAPIRE, 1997).

Hastie, Tibshirani e Friedman (2009) interpretaram esse comportamento de reponderação como um mecanismo que confere às observações mais difíceis de serem corretamente classificadas uma influência crescente ao longo das iterações. Embora o AdaBoost atue por meio da combinação sequencial de modelos e mantenha sua estrutura fixa ao longo das iterações, o processo de reponderação progressiva faz com que os classificadores subsequentes concentrem a atenção nas regiões do espaço de entrada associadas a maiores taxas de erro. Esse direcionamento sucessivo pode ser entendido como uma forma indireta de alocar mais esforço computacional às amostras mais difíceis. (HASTIE; TIBSHIRANI; FRIEDMAN, 2009)

O tratamento das amostras de maneira diferenciada conforme sua complexidade é também princípio de arquiteturas neurais dinâmicas, particularmente nas redes com saídas antecipadas (*early exits*). Nesses modelos, amostras simples podem ser classificadas precocemente, enquanto amostras complexas percorrem camadas adicionais, consumindo mais recursos computacionais. Embora AdaBoost e redes com *early exits* operem em paradigmas distintos (ensemble de modelos versus arquitetura única com múltiplos pontos de saída) ambos compartilham a

lógica central de modular o esforço computacional em função da dificuldade da entrada.

Assim, é possível estabelecer um paralelo conceitual entre o AdaBoost e as arquiteturas dinâmicas modernas. Ambos os métodos ajustam seu comportamento em resposta à complexidade dos exemplos, ainda que por mecanismos distintos. Essa convergência reforça a ideia de que estratégias adaptativas de inferência têm raízes em métodos clássicos de *ensemble* e podem ser observadas em abordagens recentes baseadas em aprendizado profundo.

2.4 EARLY EXIT EM REDES NEURAIS DINÂMICAS

O conceito de *Early Exit* (saída antecipada) em redes neurais profundas dinâmicas refere-se a uma estratégia em que a inferência pode ser interrompida antes de atingir todas as camadas do modelo, desde que uma predição intermediária satisfaça um critério predefinido. Embora seja frequentemente exemplificado em tarefas de classificação, não se restringe apenas a esse domínio, podendo ser aplicado a detecção de objetos, segmentação e outras aplicações que se beneficiem de diferentes níveis de abstração ao longo da rede(P et al., 2024)(HAN et al., 2021).

Dessa forma, a motivação central para a introdução de saídas antecipadas reside na observação de que, em grande parte das amostras, a decisão pode ser tomada de forma segura sem explorar toda a profundidade da rede, economizando recursos computacionais. A ideia de "sair" cedo é especialmente útil quando as amostras apresentadas são de baixa complexidade, o que permite que o critério de confiança ou complexidade (por exemplo, entropia, distribuição de probabilidades ou limiar pré-definido) seja atendido nas primeiras camadas. Por outro lado, entradas que exigem análises mais sofisticadas continuam até estágios avançados da rede, garantindo que a qualidade global das predições não seja comprometida. De maneira geral, esse equilíbrio entre confiança na predição antecipada e custo computacional torna o *Early Exit* uma técnica atrativa para aplicações em ambientes com recursos limitados ou que demandem inferência em tempo real.

Em trabalhos como o BranchyNet (TEERAPITTAYANON; MCDANEL; KUNG, 2017), por exemplo, cada ponto de saída intermediário é dotado de um classificador próprio que avalia se o nível de confiança atual é suficiente para encerrar a inferência; caso contrário, o processamento continua para camadas mais profundas. Essa abordagem, embora ilustrada em cenários de classificação de imagens, pode ser incorporada a arquiteturas específicas de detecção ou segmentação ao empregar *heads* intermediários especializados em cada tarefa.

Na figura 20, inspirada em Teerapittayanon et al. (2016), observa-se claramente a inser-

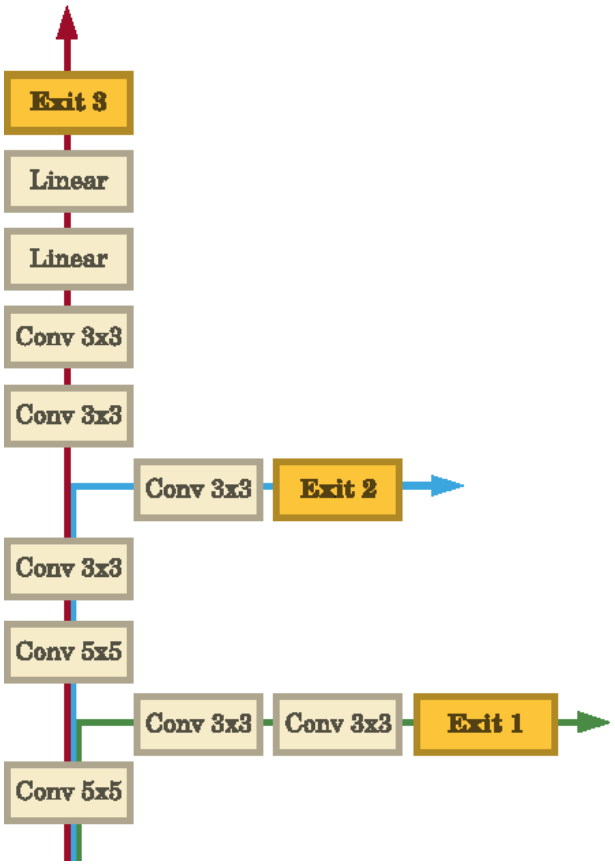


Figura 20 – Arquitetura do BranchyNet(TEERAPITTAYANON; MCDANEL; KUNG, 2017)

ção de *ramos* (*branches*) ao longo do fluxo de processamento principal da rede (mostrado na coluna central). Cada ramo fornece um ponto de saída antecipada (*early exit*) com seu próprio conjunto de camadas convolucionais e/ou lineares, formando assim "preditores intermediários"capazes de gerar respostas parciais sem percorrer toda a profundidade do modelo. O princípio de adicionar múltiplos ramos, cada um em diferentes camadas, reflete justamente a ideia central do (*early exit*): processar rapidamente amostras simples, enquanto as mais complexas prosseguem para estágios subsequentes do *backbone* convolucional.

Para exemplificar, considerando uma rede neural profunda com n saídas antecipadas $S = \{s_1, s_2, \ldots, s_n\}$. Para uma amostra de entrada $\mathbf{x}$, o modelo calcula predições $f(\mathbf{x})$ em diferentes pontos intermediários. A decisão de interrupção antecipada é formalizada como:

$$s_k = \begin{cases} s_k(f_k(\mathbf{x})) & \text{se } C_k \geq T_k \\ s_{k+1}(f_{k+1}(\mathbf{x})) & \text{caso contrário} \end{cases} \tag{2.20}$$

onde C_k representa a probabilidade (valor de confiança) atribuída pela camada *softmax* à classe majoritária na saída intermediária s_k, e T_k é um limiar de confiança pré-definido. O processo continua até que uma saída satisfaça o critério ou até que a saída final seja atingida.

2.4.1 Desafios e Limitações do Early Exit

Embora a técnica de *Early Exit* apresente benefícios evidentes, também introduz desafios técnicos e limitações que devem ser considerados. O principal desafio é a necessidade de calibrar adequadamente os limiares de confiança para cada saída intermediária. Se o limiar for muito conservador, a maioria das amostras será processada até a última camada, anulando o benefício computacional. Por outro lado, limiares excessivamente relaxados podem levar a decisões prematuras e perda de precisão.

Outro desafio importante está no treinamento da rede. Como as saídas intermediárias precisam fornecer predições confiáveis, os ramos auxiliares devem ser treinados conjuntamente com o modelo principal, o que pode gerar conflitos de gradientes. Estratégias como o treinamento em duas etapas ou o uso de *knowledge distillation* são frequentemente exploradas para mitigar esses conflitos e garantir que as saídas intermediárias aprendam representações robustas.

Além disso, a implementação de múltiplas saídas antecipadas aumenta a complexidade arquitetural do modelo, o que pode demandar ajustes em bibliotecas de aprendizado profundo e compatibilidade com hardware especializado, como GPUs otimizadas para computação condicional (P et al., 2024).

Abordagens de Treinamento para Redes Multi-Exit

O treinamento de redes neurais profundas com múltiplas saídas antecipadas (*multi-exit networks*) apresenta desafios complexos, pois envolve otimizar tanto a saída final da rede quanto as saídas intermediárias, balanceando eficiência e precisão. As metodologias abordadas incluem treinamento conjunto, treinamento por ramo, treinamento separado, treinamento em duas etapas e treinamento baseado em *knowledge distillation*(P et al., 2024) .

2.4.2 Treinamento Conjunto (Joint Training)

O **treinamento conjunto** consiste em otimizar todas as saídas da rede, tanto antecipadas quanto finais, durante o mesmo ciclo de retropropagação. A função de perda total é uma soma ponderada das perdas individuais de cada saída:

$$\mathcal{L}_{\text{total}}(\theta) = \sum_{i=1}^{n} w_i L_i(s_i(f_i(\mathbf{x})), \mathbf{y}) \tag{2.21}$$

A Equação acima descreve o treinamento de um modelo com múltiplas saídas, no qual todas as previsões – intermediárias e finais – são otimizadas conjuntamente por meio de uma única função de perda global. A função $\mathcal{L}_{\text{total}}(\theta)$ é definida como a soma ponderada das perdas individuais de cada saída i, onde w_i representa o peso atribuído a cada componente de perda, L_i é a função de perda específica para a saída i, e $s_i(f_i(x))$ corresponde à predição realizada na i-ésima saída da rede para a entrada x.

A principal vantagem dessa abordagem é a possibilidade de atualizar todos os parâmetros em uma única etapa de retropropagação, o que pode acelerar o treinamento na prática. Além disso, saídas intermediárias e final aprendem de maneira integrada, de modo que o extrator de características se beneficia dos gradientes gerados por cada *exit*, potencialmente adquirindo representações mais gerais. Por outro lado, essa configuração pode enfrentar conflitos de gradientes, pois diferentes saídas antecipadas podem exigir ajustes distintos nos pesos, dependendo do nível de complexidade da amostra que cada uma processa. Esses conflitos podem tornar a convergência mais lenta ou induzir o modelo a mínimos locais menos satisfatórios, especialmente se não houver um balanceamento cuidadoso entre os pesos. Mesmo assim, o treinamento conjunto permanece uma estratégia amplamente adotada por combinar simplicidade de implementação com ganhos de eficiência no ajuste global dos parâmetros(P et al., 2024)(HAN et al., 2021)

Algorithm 1 Treinamento Conjunto para Redes Multi-Exit

Require: Conjunto de dados $\mathcal{D}$, taxa de aprendizado η, conjunto de pesos $(w_1, w_2, .., w_n)$

1: Inicializar os parâmetros da rede θ
2: **for** cada lote $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ **do**
3: Calcular as predições de todas as saídas $\hat{\mathbf{y}}_i = s_i(f_i(\mathbf{x}))$
4: Calcular a perda total: $\mathcal{L}_{\text{total}} = \sum_{i=1}^{n} w_i L_i(\hat{\mathbf{y}}_i, \mathbf{y})$
5: Retropropagação e atualização dos pesos: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{total}}$
6: **end for**

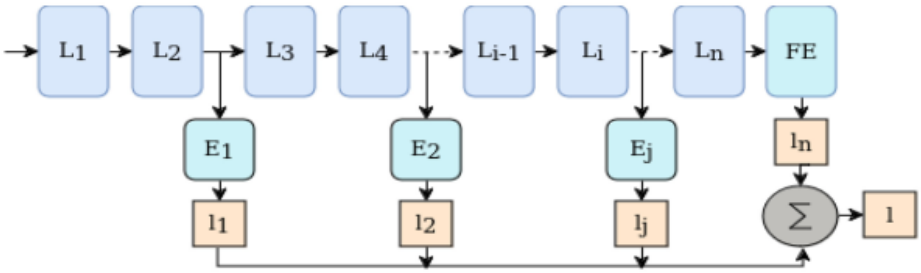


Figura 21 – Ilustração do Treinamento Conjunto(P et al., 2024).

2.4.3 Treinamento por Ramo (Branch-wise Training)

O treinamento por ramo consiste em otimizar cada *early exit* de forma sequencial, congelando os pesos já ajustados assim que uma saída antecipada atinge o desempenho desejado, antes de iniciar a otimização das próximas. Dessa forma, cada *branch* é tratado como um classificador parcial independente, recebendo gradientes apenas durante sua fase específica de treinamento. Por exemplo, se existirem duas saídas antecipadas, tem-se:

$$\mathcal{L}_1(\theta_1) = L_1\Big(s_1(f_1(\mathbf{x})), \mathbf{y}\Big), \tag{2.22}$$

$$\mathcal{L}_2(\theta_2) = L_2\Big(s_2(f_2(f_1(\mathbf{x}))), \mathbf{y}\Big), \tag{2.23}$$

onde s_1 e s_2 representam as predições das duas saídas, f_1 e f_2 são funções que retornam as representações intermediárias (features) extraídas pelo *backbone* nos pontos onde os respectivos ramos são inseridos, L_1 e L_2 são as funções de perda associadas a cada saída, e θ_1, θ_2 correspondem aos parâmetros do modelo ajustados em cada etapa.

A vantagem imediata dessa abordagem é o controle refinado sobre o ajuste de cada ramo. Ao treinar primeiramente a primeira saída e congelar seus parâmetros, assegura-se que a saída inicial seja especializada em extrair representações úteis para amostras simples, por exemplo. Em seguida, passa-se à saída intermediária ou final, que pode operar sobre características já consolidadas, sem alterar os pesos da fase anterior. Por outro lado, esse procedimento eleva o tempo de treinamento, pois cada ramo é otimizado separadamente. Além disso, a ausência de atualizações conjuntas entre as saídas pode reduzir a sinergia entre os diferentes níveis de abstração, já que gradientes distintos não são propagados simultaneamente pelas mesmas camadas.

Algorithm 2 Treinamento por Ramo para Redes Multi-Exit

Require: Conjunto de dados $\mathcal{D}$, taxa de aprendizado η

Inicializar os parâmetros da rede θ
for cada saída s_i de 1 até n **do**
    for cada lote $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ **do**
        Calcular a predição: $\hat{\mathbf{y}}_i = s_i(f_i(\mathbf{x}))$
        Calcular a perda: $\mathcal{L} = L(\hat{\mathbf{y}}_i, \mathbf{y})$
        Atualizar os pesos: $\theta_i \leftarrow \theta_i - \eta \nabla_{\theta_i} \mathcal{L}$
    end for
    Congelar os pesos da saída atual
end for

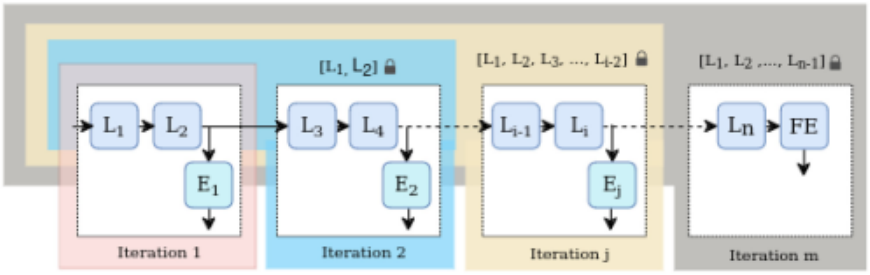


Figura 22 – Ilustração do Treinamento por Ramo(P et al., 2024).

2.4.4 Treinamento em Duas Etapas (Two-Stage Training)

O **treinamento em duas etapas** segue uma lógica intermediária entre o *treinamento conjunto* e o *treinamento por ramo*. Nessa estratégia, a rede principal (ou *backbone*) é treinada inicialmente sem considerar as saídas antecipadas. Uma vez concluído esse primeiro estágio, os parâmetros do *backbone* são congelados. Em seguida, no segundo estágio, cada *branch* lateral (saída antecipada) é treinado individualmente, reutilizando os pesos congelados das camadas iniciais.

A Figura 23 ilustra essa abordagem de forma esquemática. Observe que, assim como no *treinamento por ramo*, cada *exit* é tratado como um preditor auxiliar. A diferença fundamental aqui é que o *backbone* permanece inalterado durante o treinamento dos ramos laterais, pois seus pesos já foram previamente ajustados.

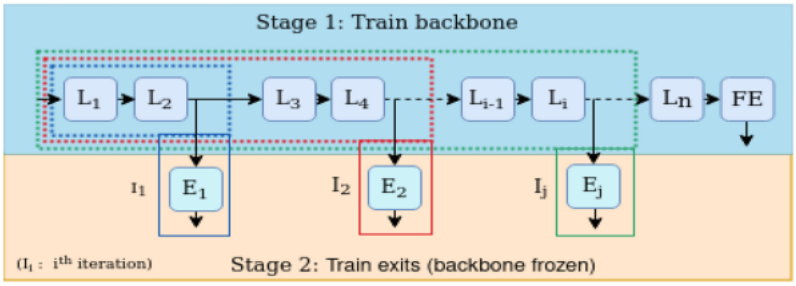


Figura 23 – Ilustração do Treinamento em Duas Etapas(P et al., 2024).

Algorithm 3 Treinamento em Duas Etapas para Redes Multi-Exit

Require: Conjunto de dados $\mathcal{D}$, taxa de aprendizado η

1: **Etapa 1:** Treinar o *backbone* sem as saídas antecipadas
   - Inicializar e treinar as camadas $L_1, L_2, \ldots, L_n$, FE (onde FE é a camada final) minimizando apenas a perda referente à saída principal.
   - Ao final dessa etapa, congelar os parâmetros do *backbone*.

2: **Etapa 2:** Para cada *branch* lateral E_i
   - Fixar (reutilizar) os pesos já aprendidos do *backbone*.
   - Treinar apenas os parâmetros específicos do *branch* E_i, minimizando a perda referente à saída antecipada $s_i(f_i(\mathbf{x}))$.

A adoção de um *backbone* previamente treinado e mantido congelado na construção de modelos de múltiplas saídas apresenta vantagens importantes, mas também traz desafios relevantes. Entre as principais vantagens, destaca-se a simplificação do ajuste dos ramos laterais, pois o *backbone* já está consolidado e permanece inalterado, o que permite converter rapidamente uma rede pré-treinada em um modelo com múltiplas saídas. Esse procedimento viabiliza a inclusão de *ramos* adicionais sem afetar a estrutura principal do modelo. Além disso, por manter as camadas iniciais fixas, a abordagem torna mais simples a experimentação com classificadores não neurais, como *SVMs* e árvores de decisão, os quais podem ser acoplados diretamente aos ramos laterais, fazendo uso das representações extraídas pelo *backbone*.

Por outro lado, essa estratégia também impõe alguns desafios. Ao congelar o *backbone*, as camadas iniciais deixam de receber as correções provenientes dos ramos, o que significa que possíveis melhorias na extração de características não são retropropagadas para essas camadas. Consequentemente, o *backbone* pré-treinado pode não estar otimizado para funcionar como base de múltiplas saídas intermediárias. (P et al., 2024)

2.5 DETECÇÃO DE OBJETOS

A detecção de objetos está entre as tarefas centrais da visão computacional, pois une duas tarefas-chave em um único procedimento: **localizar** cada objeto na imagem por meio de caixas delimitadoras (bounding boxes) e **classificar** cada uma dessas caixas em uma determinada categoria. A importância dessa tarefa reflete-se em inúmeras aplicações práticas, como vigilância por vídeo, sistemas de assistência à condução (ADAS), robótica, análise de imagens médicas, e muito mais. (JIAO et al., 2020) (CHEN; CHEN-SONG, 2022)

A seguir, aprofunda-se tanto em aspectos teóricos quanto práticos, abrangendo a formulação do problema, estratégias de redução de complexidade, métodos de regressão e classificação, funções de perda e métricas de avaliação.

Formalmente, considere um conjunto de classes

$$C = \{c_1, c_2, \ldots, c_k\},$$

e uma imagem

$$I \in \mathbb{R}^{H \times W \times 3}.$$

O objetivo da detecção de objetos é produzir um conjunto de detecções:

$$S = \{(b_i, c_i, p(c_i|b_i))\}_{i=1}^{n},$$

onde:

- $b_i = (x, y, w, h)$ ou $b_i = (x_{min}, y_{min}, x_{max}, y_{max})$ é a caixa delimitadora (*bounding boxes*), onde na notação (x, y, w, h), (x, y) representa a coordenada superior-esquerda (ou central, dependendo da convenção) e (w, h) o tamanho da caixa; já na notação $(x_{min}, y_{min}, x_{max}, y_{max})$, (x_{min}, y_{min}) são as coordenadas superiores-esquerdas e (x_{max}, y_{max}) as inferiores-direitas.
- $c_i \in C$ é a classe prevista pelo modelo para aquela caixa;
- $p(c_i|b_i)$ é a probabilidade estimada de a caixa b_i pertencer à classe c_i.

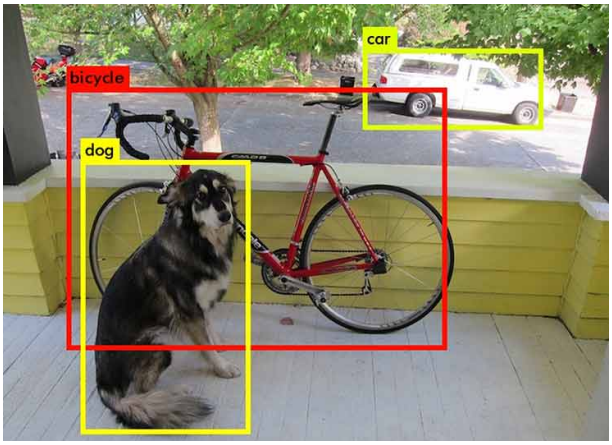


Figura 24 – Exemplo de caixas delimitadoras com classes associadas(LIU et al., 2016).

Detectar objetos "do zero" em uma imagem envolve procurar por todas as combinações de possíveis posições (x, y), larguras e alturas (w, h) e classes. Esse espaço de busca é extremamente grande, pois qualquer coordenada pode ser uma possível localização de objeto. (JIAO et al., 2020)

Por exemplo, em uma imagem $H \times W$, cada pixel (ou cada par de pixels se pensarmos em passos de varredura) pode ser um canto superior-esquerdo de uma caixa, e cada caixa pode ter tamanhos diferentes. Além disso, para cada caixa, há k possíveis classificações (mais a possibilidade de ser fundo).

Nos chamados *métodos baseados em âncoras* (*anchor-based*), como Faster R-CNN (REN et al., 2015) e SSD (LIU et al., 2016), não se faz busca exaustiva em todas as posições da imagem. Em vez disso, a busca é realizada em um conjunto fixo e gerenciável de âncoras:

1. Para cada *mapa de características* (*feature map*) de dimensão $H' \times W'$, extraído de uma rede convolucional profunda (p. ex., ResNet ou VGG).

2. Em cada célula desse mapa (cada posição (i, j)), alocam-se várias caixas *âncoras* pré-definidas, cada uma com um determinado *aspect ratio* (proporção entre largura e altura) e escala (tamanho).

3. Para cada âncora $a_k = (x_k, y_k, w_k, h_k)$, o modelo aprende *deslocamentos* (offsets) e uma pontuação de classificação.

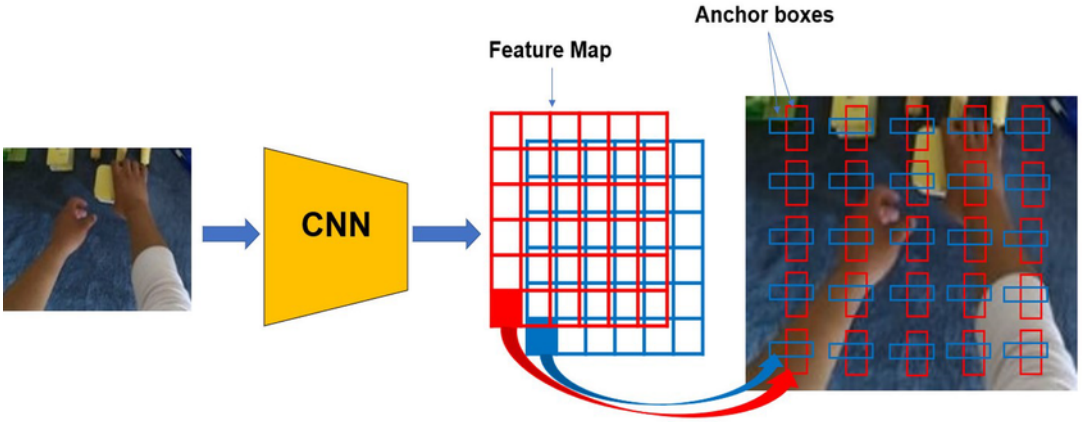


Figura 25 – Exemplo de definição de âncoras a partir de uma CNN(NGUYEN; SCHERER; LE, 2023).

Para identificar se uma âncora está próxima de uma caixa real, utiliza-se a métrica *IoU* (Intersection over Union):

$$\text{IoU}(a_k, b_i^{\text{true}}) = \frac{\text{Área}(a_k \cap b_i^{\text{true}})}{\text{Área}(a_k \cup b_i^{\text{true}})}.$$

Durante o treinamento, define-se um limiar (por exemplo, $0,5$) para classificar âncoras com IoU maior do que esse limiar como "positivas" (indicando que aquele local provavelmente contém um objeto), enquanto âncoras com IoU baixo são "negativas" (fundo).

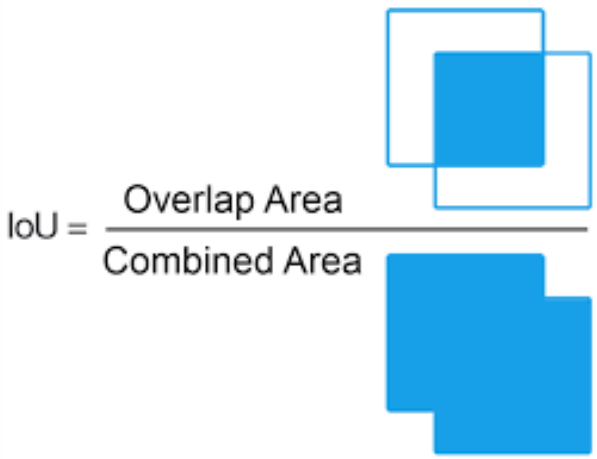


Figura 26 – Representação visual do IOU. (GRANDHE, 2021)

2.5.1 Feature Pyramid Networks (FPNs)

Um grande desafio é lidar com *objetos de diferentes escalas*. Métodos como FPN (LIN et al., 2017a) extraem mapas de características em diferentes níveis de profundidade da rede (geralmente de uma *backbone* como ResNet-50, ResNet-101 etc.). Cada nível da pirâmide lida melhor com uma faixa de tamanho de objeto:

- **Níveis altos da rede** (features profundas): Boa capacidade semântica, mas resolução espacial reduzida — úteis para objetos maiores.
- **Níveis mais baixos** (features menos profundas): Maior resolução espacial, mas características menos complexas — úteis para objetos pequenos.

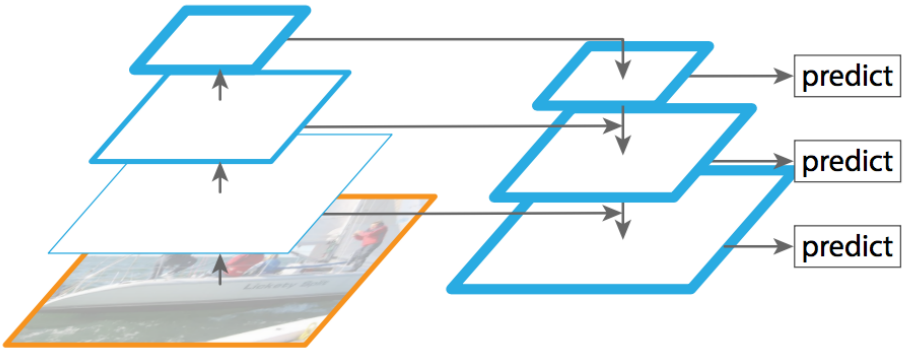


Figura 27 – Ilustração da lógica de uma FPN. (LIN et al., 2017b)

2.5.2 Regressão e Classificação Conjuntas

Em detecção de objetos, cada âncora (ou cada região proposta) passa por *duas previsões*: (i) uma *regressão* para ajustar a caixa delimitadora e (ii) uma *classificação* para determinar a classe do objeto (ou se é fundo).

2.5.3 Regressão de Caixas

A regressão procura alinhar cada âncora $a_k = (x_k, y_k, w_k, h_k)$ à caixa real $b_i^{\text{true}} = (x, y, w, h)$. Para tanto, define-se um conjunto de deslocamentos:

$$\Delta x = \frac{x - x_k}{w_k}, \quad \Delta y = \frac{y - y_k}{h_k},$$

$$\Delta w = \log\Big(\frac{w}{w_k}\Big), \quad \Delta h = \log\Big(\frac{h}{h_k}\Big).$$

No momento de inferência, a rede prevê $(\hat{\Delta}x, \hat{\Delta}y, \hat{\Delta}w, \hat{\Delta}h)$. Então, a *caixa predita* é dada por:

$$\hat{x} = x_k + \hat{\Delta}x \cdot w_k, \quad \hat{y} = y_k + \hat{\Delta}y \cdot h_k,$$

$$\hat{w} = w_k \cdot \exp(\hat{\Delta}w), \quad \hat{h} = h_k \cdot \exp(\hat{\Delta}h).$$

2.5.4 Classificação

Cada âncora também recebe uma pontuação de *probabilidade* para cada classe $c \in C$. Comumente, um classificador (geralmente uma camada totalmente conectada seguida de uma ativação *softmax*) retorna:

$$p(c|a_k) = \frac{\exp(W_c \cdot \phi(a_k) + b_c)}{\sum_{j=1}^{k} \exp(W_j \cdot \phi(a_k) + b_j)},$$

A Equação define a probabilidade $p(c \mid a_k)$ de que a âncora a_k pertença à classe c. Essa probabilidade é calculada a partir de uma camada totalmente conectada seguida de uma ativação *softmax*, que converte os logits em uma distribuição de probabilidade sobre todas as classes. Na equação, $\phi(a_k)$ representa o vetor de *features* extraídas pela rede para a âncora a_k; W_c e b_c são os parâmetros (pesos e viés) do classificador para a classe c. O numerador $\exp(W_c \cdot \phi(a_k) + b_c)$ corresponde à pontuação não normalizada para a classe c, enquanto o denominador executa a soma sobre todas as classes $j = 1, \ldots, k$, assegurando a normalização. Dessa forma, a equação implementa uma classificação multiclasse padrão via *softmax*, aplicada às representações aprendidas para cada âncora.

A interdependência entre regressão e classificação é fundamental: se a caixa estiver mal ajustada, as características extraídas podem ficar desalinhadas em relação ao objeto, levando a uma pior performance de classificação, e vice-versa.

2.5.5 Função de Perda Conjunta

Treina-se a rede de detecção por meio de uma combinação de perdas de *classificação* e *regressão*, tipicamente somadas:

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \lambda\, \mathcal{L}_{\text{reg}}.$$

1. **Perda de Classificação** ($\mathcal{L}_{\text{cls}}$)

   Geralmente, utiliza-se a *entropia cruzada* (cross-entropy), como em classificadores tradicionais. Algumas variantes empregam *focal loss* (RetinaNet (LIN et al., 2017c)) para lidar com desbalanceamento (objetos vs. fundo).

2. **Perda de Regressão** ($\mathcal{L}_{\text{reg}}$)

   Avalia a proximidade entre os deslocamentos preditos $(\hat{\Delta}x, \hat{\Delta}y, \hat{\Delta}w, \hat{\Delta}h)$ e os deslocamentos reais $(\Delta x, \Delta y, \Delta w, \Delta h)$. Usa-se com frequência a *Smooth L1* (Huber Loss) ao invés de L1 puro, pois ela penaliza menos discrepâncias pequenas e estabiliza o treinamento.

A constante λ equilibra a ênfase entre a classificação e a regressão, já que erros de classificação são de natureza diferente dos erros de localização.

2.5.6 Pós-Processamento: *Non-Maximum Suppression* (NMS)

Após prever caixas e probabilidades para cada âncora, muitas dessas caixas podem se sobrepor fortemente, resultando em *múltiplas detecções* para o mesmo objeto. Se não houver um mecanismo para filtrar as previsões, o modelo pode retornar diversas caixas sobrepostas com pontuações semelhantes, tornando os resultados confusos e difíceis de interpretar.

O NMS resolve esse problema removendo caixas redundantes e mantendo apenas aquelas com pontuações mais altas, garantindo que cada objeto tenha apenas uma única caixa delimitadora.

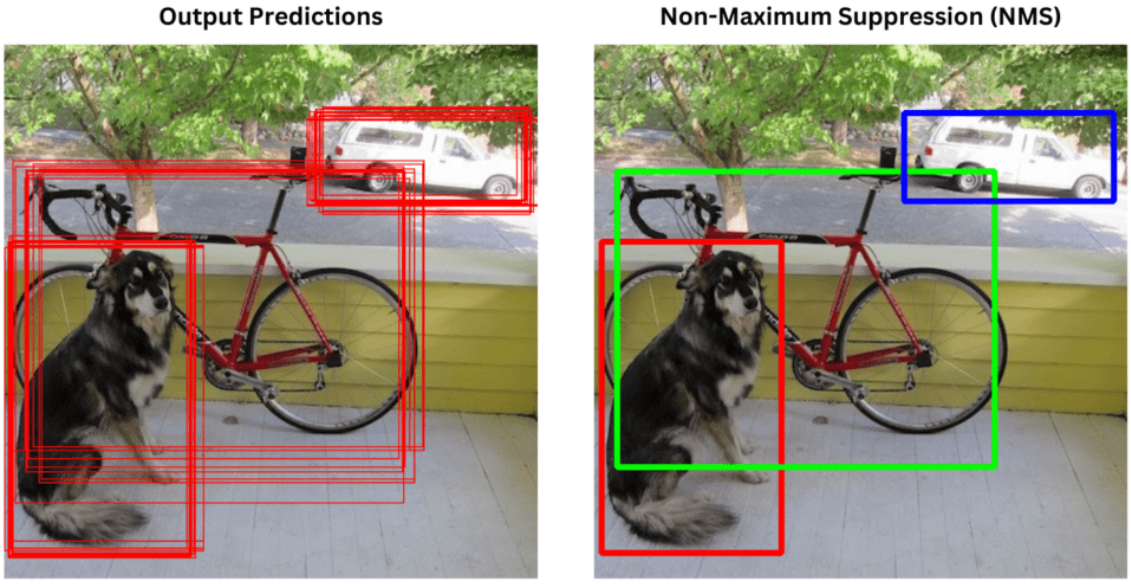


Figura 28 – Exemplo de aplicação do NMS(ULTRALYTICS, 2024)

Algorithm 4 *Non-Maximum Suppression* (NMS)

Require: $\mathcal{B}$: Lista de caixas delimitadoras
Require: $\mathcal{S}$: Lista de scores correspondentes
Require: τ: Threshold de IoU
Ensure: $\mathcal{B}^*$: Lista de caixas selecionadas
Ordenar $\mathcal{B}$ por $\mathcal{S}$ em ordem decrescente
Inicializar $\mathcal{B}^* \leftarrow \emptyset$
while $\mathcal{B}$ não estiver vazia **do**
    Selecionar b_{best} da $\mathcal{B}$ com maior score
    Adicionar b_{best} à lista $\mathcal{B}^*$
    for cada b_i em $\mathcal{B}$ **do**
        Calcular $IoU(b_{\text{best}}, b_i)$
        if $IoU(b_{\text{best}}, b_i) > \tau$ **then**
            Remover b_i de $\mathcal{B}$
        end if
    end for
end while
return $\mathcal{B}^*$

2.5.7 Métricas de Avaliação: AP e mAP

AP (Average Precision):

Para cada classe, constrói-se a *curva de Precisão vs. Revocação* (Precision-Recall). A AP corresponde, essencialmente, à área sob essa curva. Pode ser calculada de forma discreta,

somando-se precisões em pontos de revocação:

$$\text{AP}_c = \sum_{i=1}^{m} P(r_i)\,\Delta r_i,$$

onde $P(r_i)$ é a precisão na revocação r_i, e $\Delta r_i = r_i - r_{i-1}$.

mAP (mean Average Precision):

É a média das APs para todas as classes:

$$\text{mAP} = \frac{1}{k} \sum_{c=1}^{k} \text{AP}_c.$$

Para avaliar a robustez do modelo em diferentes níveis de rigor de sobreposição, utiliza-se o *mAP@[0.5:0.95]*, adotado no *COCO Dataset* (LIN et al., 2014). Ele calcula a média do mAP em incrementos de 0.05, desde IoU=0.5 até 0.95:

$$\text{mAP}[0.5{:}0.95] = \frac{1}{10} \sum_{i=0}^{9} \text{mAP@IoU}\Big(0.5 + 0.05 \cdot i\Big).$$

Isso gera uma avaliação mais completa, pois um modelo pode ter bom desempenho em limiares baixos (p. ex., IoU=0.5) mas falhar quando se exige maior sobreposição.

O Single Shot MultiBox Detector (SSD) (LIU et al., 2016) é um modelo de detecção de objetos *one-stage* que realiza, em um único processo de inferência, tanto a predição das localizações quanto a atribuição de classes para cada objeto na cena. Esse modelo se alicerça em uma arquitetura convolucional de base, frequentemente derivada de redes como a VGG-16, a qual é responsável pela extração dos mapas de características iniciais. Após essa primeira etapa de extração de características, o SSD adiciona camadas convolucionais suplementares, projetadas para operar em escalas espaciais progressivamente menores, funcionando como uma FPN. Cada uma dessas camadas gera mapas de ativação específicos, em que se posicionam as chamadas *default boxes* (ou *prior boxes*), que são caixas pré-definidas com diferentes proporções e escalas. A rede, então, aprende a prever deslocamentos (offsets) e pontuações de classe para cada default box, dispensando qualquer estágio intermediário de seleção ou refinamento de propostas.

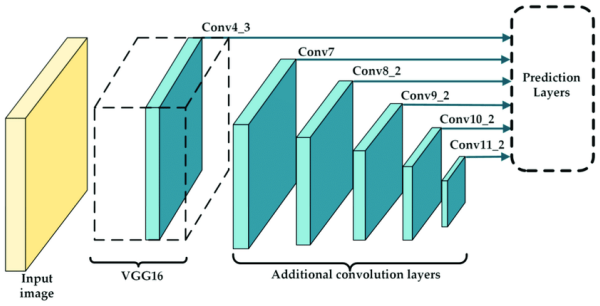
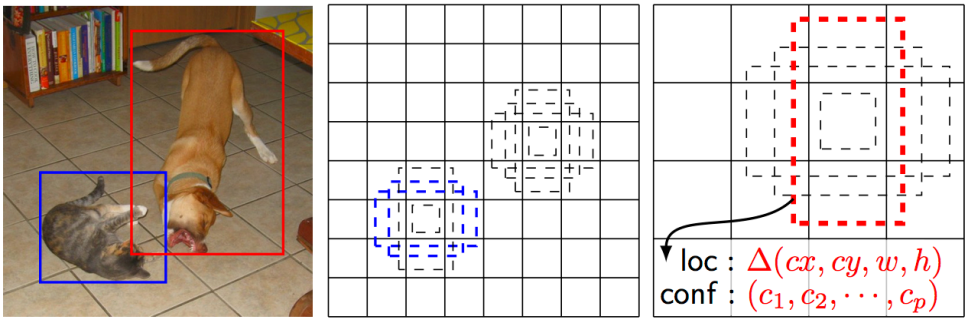


Figura 29 – Ilustração do SSD original com VGG16 como *backbone*. (LIU et al., 2016)

Em termos de formalização, considere uma rede base $\mathcal{B}$ que, partindo de uma imagem de entrada $\mathbf{I} \in \mathbb{R}^{H\times W\times 3}$, produz mapas de características convolucionais $\mathbf{\Phi}_k$ em diversos níveis. Esses mapas alimentam camadas adicionais, $\{\Psi_1, \Psi_2, \ldots, \Psi_L\}$, que, por sua vez, geram saídas de dimensão reduzida, $\mathbf{A}_\ell \in \mathbb{R}^{H_\ell\times W_\ell\times D_\ell}$, para $\ell \in \{1, \ldots, L\}$. Em cada posição (i, j) de $\mathbf{A}_\ell$, definem-se caixas pré-definidas de índices $d \in \{1, \ldots, m_\ell\}$, totalizando m_ℓ caixas por posição. Cada default box b_{ij}^d é parametrizada por $(x_{ij}^d,\, y_{ij}^d,\, w_{ij}^d,\, h_{ij}^d)$, onde (x_{ij}^d, y_{ij}^d) são as coordenadas normalizadas de seu centro e (w_{ij}^d, h_{ij}^d) representam, respectivamente, a largura e a altura, também em valores normalizados em função das dimensões da imagem original. Esses parâmetros são definidos com base em escalas e proporções estabelecidas de antemão, distribuídas ao longo das camadas para cobrir objetos de múltiplos tamanhos.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figura 30 – Fluxo de representação de detecção de objetos no SSD (LIU et al., 2016). (a) A imagem original com caixas delimitadoras do ground truth (GT) para os objetos presentes. (b) Um mapa de características 8×8, onde diferentes áreas da imagem são representadas e as caixas delimitadoras são mapeadas em escalas apropriadas. (c) Um mapa de características 4×4, onde os ajustes de localização (loc: $\Delta(cx, cy, w, h)$) e classificações de confiança ($conf$: $(c_1, c_2, \ldots, c_p)$) são gerados para objetos detectados em resoluções menores.

O processo de regressão consiste em aprender, para cada default box, uma correção que aproxime suas coordenadas às do objeto verdadeiro. Formalmente, se $b^{\text{true}} = (x^{\text{true}}, y^{\text{true}}, w^{\text{true}}, h^{\text{true}})$ é a caixa real associada àquela posição, define-se um vetor de deslocamentos $\Delta = (\Delta x,\, \Delta y,\, \Delta w,\, \Delta h)$ a partir do desvio entre $\left(x^{\text{true}}, y^{\text{true}}, w^{\text{true}}, h^{\text{true}}\right)$ e $\left(x_{ij}^d, y_{ij}^d, w_{ij}^d, h_{ij}^d\right)$. Comumente, empregam-se as transformações normalizadas

$$\Delta x = \frac{x^{\text{true}} - x_{ij}^d}{w_{ij}^d}, \quad \Delta y = \frac{y^{\text{true}} - y_{ij}^d}{h_{ij}^d}, \quad \Delta w = \log\!\left(\frac{w^{\text{true}}}{w_{ij}^d}\right), \quad \Delta h = \log\!\left(\frac{h^{\text{true}}}{h_{ij}^d}\right).$$

O modelo prevê $\hat{\Delta} = (\hat{\Delta}x,\, \hat{\Delta}y,\, \hat{\Delta}w,\, \hat{\Delta}h)$ e, durante a inferência, reverte-se a parametrização para recuperar a caixa final estimada, com coordenadas

$$\hat{x} = x_{ij}^d + \alpha\, \hat{\Delta}x\, w_{ij}^d, \quad \hat{y} = y_{ij}^d + \alpha\, \hat{\Delta}y\, h_{ij}^d, \quad \hat{w} = w_{ij}^d \exp(\beta\, \hat{\Delta}w), \quad \hat{h} = h_{ij}^d \exp(\beta\, \hat{\Delta}h),$$

sendo α e β fatores de calibração que limitam a magnitude dos deslocamentos.

A predição de classe se realiza paralelamente, por meio de uma camada convolucional que produz, para cada posição (i, j) e índice d, probabilidades $\mathbf{p}_{ij}^{d} = (p_{ij}^{d,1}, p_{ij}^{d,2}, \ldots, p_{ij}^{d,k}, p_{ij}^{d,bg})$ referentes às k classes consideradas e à categoria "fundo". Em geral, adota-se uma ativação $\mathrm{softmax}$ para que $\sum_{c=1}^{k} p_{ij}^{d,c} + p_{ij}^{d,bg} = 1$. Como resultado, cada default box recebe simultaneamente um vetor de deslocamentos para regressão e um vetor de probabilidades de classe. O volume de predições brutas pode alcançar dezenas de milhares de caixas por imagem, dependendo do número de camadas de detecção e da densidade de caixas em cada mapa.

Para conduzir o treinamento e associar cada default box a uma anotação de objeto real, o SSD utiliza um critério de emparelhamento baseado em Intersection over Union (IoU). Concretamente, define-se um limiar θ, e toda default box cuja IoU com uma caixa real exceda θ é marcada como positiva para aquela classe; caso não exceda esse valor para nenhuma caixa real, a box é rotulada como negativa (fundo). Também se aplica um emparelhamento reverso para garantir que cada objeto real seja associado à box que melhor o representa. Em seguida, a função de perda soma um termo de classificação, normalmente a entropia cruzada entre as probabilidades preditas e o rótulo $\mathbf{p}^{\mathrm{true}}$, e um termo de localização (regressão), que compara $\hat{\Delta}$ com Δ^{true}, normalmente via Smooth L1 ou L1. O treinamento busca minimizar essa perda global, ponderada por um fator que equilibra o erro de classificação e o de regressão, normalmente normalizada pelo número de boxes positivas.

Ao término da predição, as predições redundantes passam por um limiar de confiança para descartar caixas com baixa probabilidade e, em seguida, por uma etapa de Non-Maximum Suppression (NMS), onde suprime-se qualquer predição cuja IoU seja muito elevada em relação a outra detecção de maior confiança. Dessa forma, o SSD consolida um conjunto final de detecções. O caráter "single shot" provém do fato de a inferência coletar simultaneamente, em cada camada, as possíveis posições e rótulos dos objetos, sem exigir a prévia geração de propostas de regiões. Com isso, há um ganho de velocidade considerável em comparação aos detectores two-stage, ao mesmo tempo em que o uso de múltiplas escalas nas camadas de detecção permite lidar razoavelmente bem com objetos de tamanhos distintos. Em síntese, o arcabouço do SSD combina a engenharia cuidadosa de caixas pré-definidas, a regressão de offsets e a classificação em múltiplos níveis de resolução, resultando em um fluxo direto, de passagem única, e consistente com a necessidade de detecção rápida e acurada em domínios diversos.

3 TRABALHOS RELACIONADOS

A busca por eficiência computacional em redes neurais profundas tem motivado a exploração de arquiteturas dinâmicas que ajustem seu comportamento com base na complexidade das entradas. Essa abordagem é particularmente relevante em tarefas de detecção de objetos, que apresentam desafios significativos devido à variabilidade das características visuais, como oclusões, escalas e condições de iluminação. Simultaneamente, a avaliação da complexidade das imagens tem se mostrado uma ferramenta essencial para guiar decisões adaptativas em sistemas dinâmicos, permitindo alocar recursos computacionais de forma mais eficiente.

Nesta seção, são apresentados os principais trabalhos relacionados a detectores dinâmicos, com foco em estratégias baseadas em saídas antecipadas (*early exits*) e políticas adaptativas. Além disso, revisamos estudos que abordam a avaliação da dificuldade de imagens, destacando as contribuições metodológicas e as lacunas ainda existentes. Essa análise fornece o contexto necessário para situar a abordagem proposta neste trabalho, que integra conceitos de complexidade de imagem e detecção dinâmica em um único framework.

3.1 AVALIAÇÃO DE COMPLEXIDADE DE AMOSTRAS EM DETECÇÃO

Estimar a dificuldade em tarefas de busca visual é um campo de pesquisa multidisciplinar que combina aspectos de percepção, atenção e aprendizado de máquina. Ao longo dos anos, diversos estudos avançaram significativamente o entendimento e a modelagem deste tema, cada um contribuindo com abordagens e resultados específicos que se complementam em uma visão integrada do problema.

Tian et al. apresentaram uma abordagem computacional para estimar a dificuldade de consultas visuais com base em erros de reconstrução da consulta (TIAN et al., 2014). O método proposto utiliza registros de motores de busca e resultados de pesquisa visual para prever dificuldades e identificar padrões associados a consultas desafiadoras. Essa abordagem estabeleceu um modelo preditivo eficiente, ressaltando a relevância de dados empíricos na melhoria dos sistemas de busca visual.

Paralelamente, Hooge e Erkelens exploraram a adaptabilidade do sistema visual humano, particularmente a duração das fixações oculares em tarefas de busca visual (HOOGE; ERKELENS, 1998). Os autores demonstraram que o sistema visual ajusta dinamicamente a duração da

fixação com base nas demandas discriminatórias da tarefa, sugerindo um mecanismo subjacente de regulação temporal que equilibra esforço cognitivo e eficiência perceptual.

De uma perspectiva complementar, Bruce e Tsotsos aplicaram conceitos da teoria da informação para analisar a relação entre saliência, atenção e busca visual (BRUCE; TSOTSOS, 2009). Sua pesquisa mostrou que a relevância percebida de uma região visual influencia diretamente o desempenho em tarefas desafiadoras, evidenciando a interação entre elementos visuais e a alocação de recursos atencionais. Essa abordagem ressaltou o papel da saliência como um mediador crítico da dificuldade percebida.

Pomplun et al. investigaram o "span" visual em buscas comparativas, revelando como a dificuldade da tarefa e a atenção dividida afetam a eficiência da busca visual (POMPLUN; REINGOLD; SHEN, 2001). Esse estudo foi um dos primeiros a quantificar como fatores contextuais, como a carga cognitiva, limitam a capacidade de busca visual, fornecendo um arcabouço teórico para entender o impacto da dificuldade em ambientes complexos.

Wolfe revisou metodologias para quantificar a dificuldade em tarefas de busca visual (WOLFE, 2015). Seu trabalho abrangeu desde cenários simples até contextos aplicados, como a detecção de câncer em mamografias. Essa revisão destacou a relevância da dificuldade de busca para tarefas críticas, sugerindo que a eficácia do diagnóstico visual está intimamente ligada à complexidade do estímulo e ao treinamento do observador.

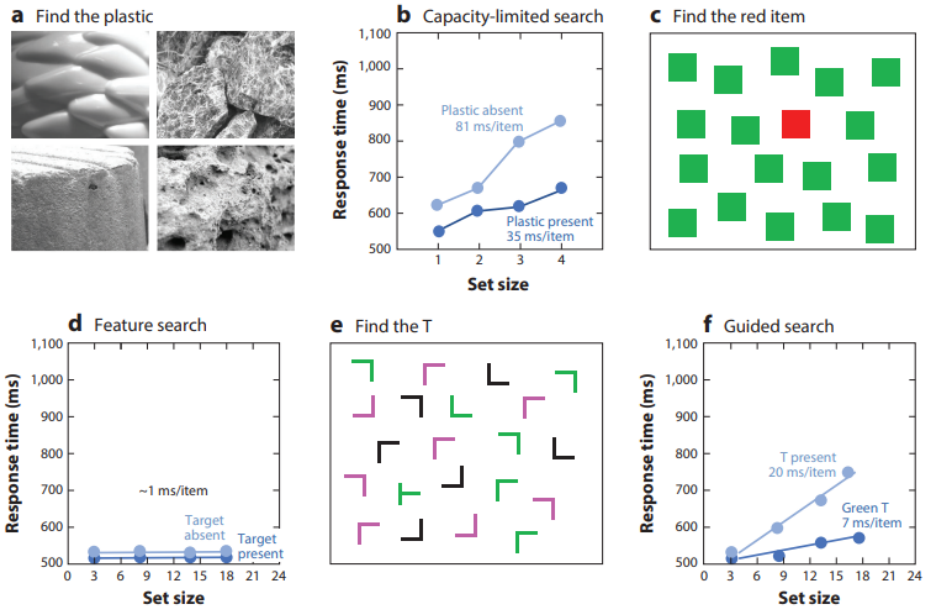


Figura 31 – Experimentos clássicos de busca visual. Cada subfigura demonstra um tipo diferente de busca visual, variando de tarefas simples a complexas, destacando os desafios de detecção em cenários visuais. Adaptado de Wolfe (2015).

Avançando para medidas fisiológicas, Porter e Troscianko introduziram a pupilometria como ferramenta para avaliar o esforço cognitivo em tarefas de busca visual e contagem (PORTER; TROSCIANKO, 2007). Seus resultados mostraram que a dilatação da pupila se correlaciona fortemente com a dificuldade percebida, abrindo novas possibilidades para a medição em tempo real da carga cognitiva.

Duchowski et al. exploraram o uso de microsacadas para estimar a dificuldade de tarefas de busca visual em superfícies em camadas (DUCHOWSKI et al., 2019). Eles constataram que mudanças na frequência de microsacadas estão diretamente associadas ao aumento da complexidade da tarefa, fornecendo uma métrica precisa para prever níveis de dificuldade em tempo real.

Rensink e Enns investigaram como o agrupamento visual de baixo nível afeta o desempenho em busca visual (RENSINK; ENNS, 1995). Eles demonstraram que elementos visualmente agrupados podem facilitar ou dificultar a busca, dependendo da semelhança entre os distratores, sugerindo que padrões de agrupamento visual influenciam significativamente o processamento visual.

Por fim, Duncan e Humphreys apresentaram uma análise detalhada de como a similaridade de estímulos afeta a dificuldade em tarefas de busca visual (DUNCAN; HUMPHREYS, 1989). Seu estudo propôs que os princípios da dificuldade de busca visual são consistentes em materiais simples e complexos, fornecendo uma base unificadora para pesquisas futuras.

Ionescu et al. (2016) propuseram uma abordagem inovadora que combina anotação humana e aprendizado profundo para estimar a dificuldade de busca em imagens (IONESCU et al., 2016). O estudo define dificuldade como o tempo de resposta necessário para localizar alvos em uma imagem. Utilizando anotações humanas coletadas por meio de crowdsourcing, os autores rotularam um subconjunto do conjunto de dados PASCAL VOC 2012 com escores de dificuldade, criando um banco de dados de referência para pesquisas futuras. Além disso, o trabalho analisou propriedades interpretáveis das imagens, como densidade de objetos, complexidade do plano de fundo e similaridade entre distratores, as quais influenciam a dificuldade percebida. Com base nessas propriedades, os autores desenvolveram um modelo de regressão treinado com recursos profundos extraídos de redes neurais convolucionais. O modelo alcançou alta precisão, classificando corretamente 75 porcento dos pares de imagens em relação à dificuldade relativa. A capacidade de generalização do modelo foi validada em classes de objetos não vistas durante o treinamento, demonstrando sua robustez. Os escores de dificuldade previstos também foram aplicados com sucesso em tarefas como localização de objetos com

supervisão fraca e classificação semissupervisionada, resultando em melhorias de desempenho.

Class	Score	mAP	Class	Score	mAP
bird	3.081	92.5%	bicycle	3.414	90.4%
cat	3.133	91.9%	boat	3.441	89.6%
aeroplane	3.155	95.3%	car	3.463	91.5%
dog	3.208	89.7%	bus	3.504	81.9%
horse	3.244	92.2%	sofa	3.542	68.0%
sheep	3.245	82.9%	bottle	3.550	54.4%
cow	3.282	76.3%	tv monitor	3.570	74.4%
motorbike	3.355	86.9%	dining table	3.571	74.9%
train	3.360	95.5%	chair	3.583	64.1%
person	3.398	95.2%	potted plant	3.641	60.7%

Figura 32 – (IONESCU et al., 2016) Também avaliou a dificuldade por classes.

3.2 DETECTORES DINAMICOS

O desafio de equilibrar precisão e eficiência tem sido amplamente estudado no contexto de redes neurais profundas. Soviany e Ionescu (2018) abordaram essa questão propondo um método que aproveita a predição da dificuldade da imagem para otimizar o uso de detectores de estágio único e de dois estágios em tarefas de detecção de objetos (SOVIANY; IONESCU, 2018). Detectores de estágio único, como o YOLO, são conhecidos por sua alta velocidade, porém frequentemente sacrificam precisão. Em contraste, detectores de dois estágios, como o Faster R-CNN, fornecem maior acurácia ao custo de tempos de processamento mais longos. Para lidar com essa compensação, os autores desenvolveram uma metodologia que incorpora a predição da dificuldade da imagem como critério para selecionar, de forma dinâmica, o tipo de detector mais adequado.

A metodologia proposta compreende duas etapas principais. Primeiro, uma rede neural convolucional é utilizada para estimar a dificuldade de uma imagem com base em fatores como densidade de objetos, oclusões e variabilidade estrutural. Esse modelo de predição é treinado em um conjunto de dados anotado, em que cada imagem recebe uma pontuação de dificuldade derivada de características extraídas automaticamente. A partir dessa pontuação, as imagens são encaminhadas para detectores de estágio único ou de dois estágios. Imagens classificadas como de baixa dificuldade são processadas por detectores de estágio único, priorizando a eficiência, enquanto imagens mais complexas são roteadas para detectores de dois estágios,

garantindo maior precisão.

Algorithm 5 Metodologia Proposta por (SOVIANY; IONESCU, 2018)

Require: I: Imagem de teste
Require: $D_{\text{rápido}}$: Detector de objeto rápido (single-stage)
Require: D_{lento}: Detector de objeto lento (two-stage)
Require: P: Preditor de dificuldade da imagem
Require: t: Limiar para separar imagens fáceis ou difíceis
Ensure: B: Conjunto de *bounding boxes* preditas
Cálculo:
if $P(I) \leq t$ **then**
    $B \leftarrow D_{\text{rápido}}(I)$
else
    $B \leftarrow D_{\text{lento}}(I)$
end if
return B

Os resultados experimentais apresentados no estudo demonstram que essa abordagem adaptativa reduz significativamente o tempo médio de inferência em diversos cenários, sem comprometer a acurácia geral. Especificamente, os autores mostram que a integração da predição de dificuldade aumenta a eficiência do sistema, evitando o uso desnecessário de detectores mais lentos para imagens mais simples e mantendo a robustez em casos complexos. Essa solução representa um avanço metodológico crucial ao combinar predições dinâmicas com a capacidade de ajustar recursos computacionais de acordo com as demandas da tarefa.

Hector et al. (2021) apresentaram uma abordagem escalável para detecção de objetos em ambientes de computação de borda, nos quais recursos computacionais são limitados e as condições operacionais podem variar significativamente (HECTOR; YUAN; LEE, 2021). A metodologia utiliza redes neurais elásticas (ENNs), combinando saídas intermediárias (early exits) com técnicas de poda de redes (network pruning) para reduzir a complexidade computacional e se adaptar dinamicamente às limitações de recursos.

O modelo baseia-se no Single Shot Detector (SSD), uma estrutura amplamente utilizada para detecção de objetos. A rede foi modificada para incluir múltiplos pontos de saída intermediária ao longo de sua arquitetura backbone, utilizando a VGG-16. Esses pontos de saída intermediária permitem que a inferência seja concluída antecipadamente, antes de todas as camadas da rede, caso as características extraídas sejam suficientes para produzir resultados confiáveis. Cada ponto de saída é seguido por uma sub-rede responsável pela localização e classificação de objetos, treinada de forma independente para maximizar a acurácia dentro das restrições de recursos.

Além disso, os autores integraram a abordagem ENN com técnicas de poda de rede, removendo pesos menos importantes com base na norma L1. Essa integração ajuda a mitigar o custo computacional adicional introduzido pelas saídas intermediárias, mantendo a precisão do modelo.

Os resultados experimentais demonstraram que a rede elástica proposta oferece uma relação custo-benefício superior em termos de acurácia e velocidade de inferência quando comparada a redes convencionais. Contudo, vale destacar que o modelo não foi analisado em um cenário real, no qual uma política ativa determina a posição de execução em tempo de execução (rpo). A ausência de tal política dinâmica na avaliação deixa questões em aberto sobre a integração prática do modelo com sistemas externos de tomada de decisão ou heurísticas que ajustem a rpo de forma adaptativa com base em condições de tempo real, como complexidade da entrada, disponibilidade de recursos ou prioridades operacionais. Essa limitação sugere áreas potenciais para pesquisa futura, especialmente no projeto e na avaliação de mecanismos adaptativos de seleção da rpo em implantações reais.

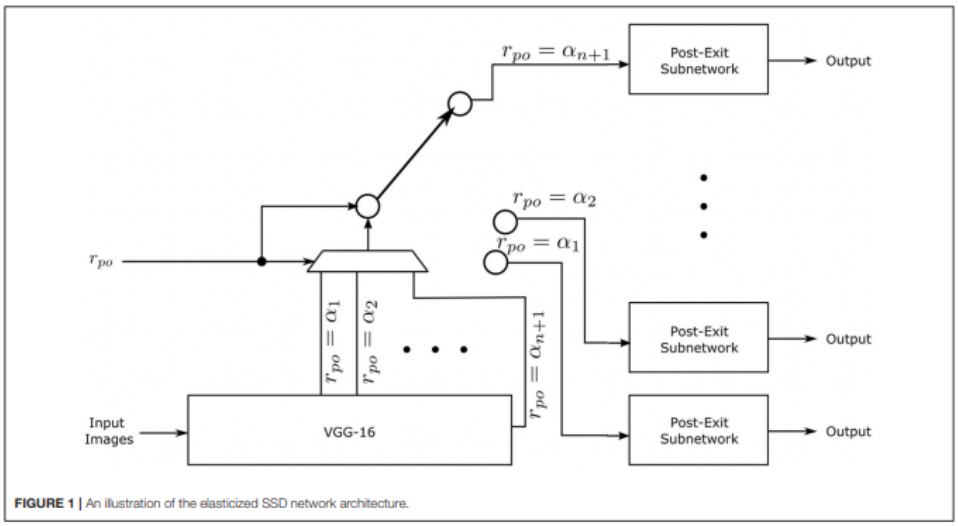


Figura 33 – Arquitetura do SSD elástico proposto em (HECTOR; YUAN; LEE, 2021), baseada em VGG-16, com múltiplos pontos de saída intermediária. Cada saída é seguida por uma sub-rede de detecção, permitindo que a inferência seja concluída antecipadamente. A decisão do ponto efetivo de saída em tempo de execução (*runtime position of execution* – rpo) ajusta dinamicamente o *trade-off* entre velocidade e acurácia, adaptando o processamento às limitações de recursos disponíveis em ambientes de borda.

Yang et al. (2024) apresentaram o AdaDet, um framework adaptativo de detecção de objetos projetado para lidar com os desafios da detecção eficiente de objetos utilizando redes com saídas antecipadas (early-exit) (YANG et al., 2024). O framework aloca recursos dinamicamente

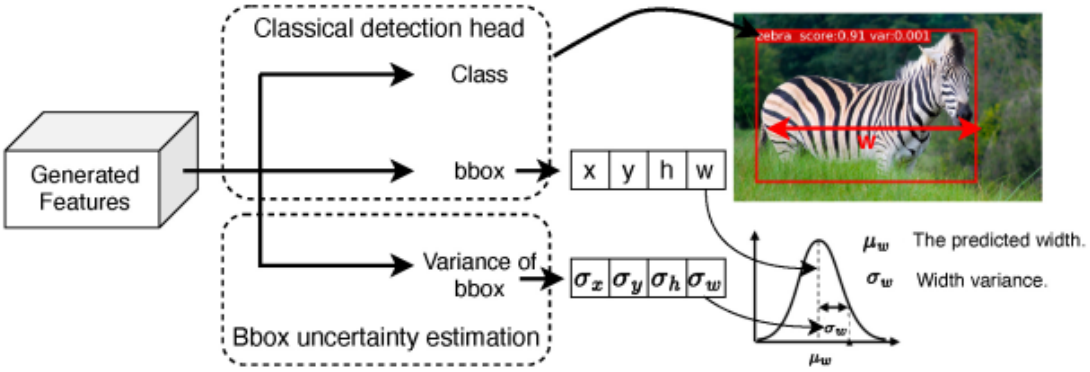


Figura 34 – Ilustração da modelagem gaussiana de *bounding boxes* proposta por (YANG et al., 2023)

com base na complexidade do input, melhorando tanto a eficiência quanto o desempenho. O sistema integra técnicas inovadoras para enfrentar dois desafios-chave na detecção adaptativa: estimar a incerteza das predições de bounding boxes e avaliar a qualidade dos resultados de detecção em imagens com múltiplos objetos.

Para abordar esses desafios, o AdaDet incorpora uma abordagem de modelagem Gaussiana para as predições de bounding boxes. Ao tratar as dimensões das caixas delimitadoras como variáveis aleatórias seguindo distribuições Gaussianas, o framework calcula intervalos de confiança para avaliar a incerteza de reconhecimento e localização. Isso possibilita uma estimativa mais detalhada da confiabilidade da detecção de objetos do que detectores convencionais, que dependem apenas de saídas determinísticas de bounding boxes.

O AdaDet também introduz um critério baseado em entropia para avaliar a qualidade geral dos resultados de detecção em uma imagem. Essa abordagem analisa a distribuição dos escores dos objetos, identificando imagens "bem detectadas" como aquelas com escores polarizados—ou seja, de alta ou baixa confiança—com pouca incerteza. Imagens que atendem a esses critérios saem da rede de forma antecipada, enquanto as mais complexas continuam a passar por camadas adicionais de processamento. A métrica de entropia é comparada a um limiar controlado por , determinando se o processamento pode ser encerrado antecipadamente ou se a imagem deve prosseguir para camadas adicionais em busca de maior precisão.

A arquitetura do AdaDet emprega a MSDNet como rede *backbone*, garantindo extração robusta de recursos e reduzindo a interferência entre cabeças de detecção. Cabeças de detecção intermediárias são acopladas estrategicamente a camadas iniciais do *backbone*, permitindo que a rede processe imagens simples com menor custo computacional. Durante o treinamento, as

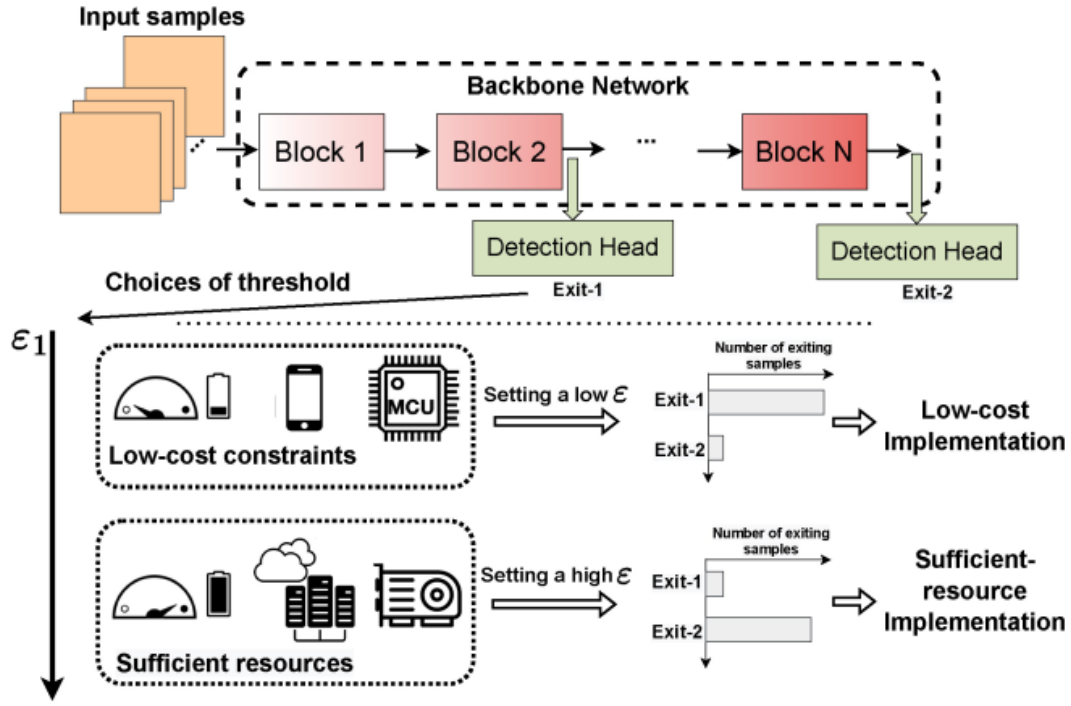


Figura 35 – Implementação do AdaDet em distintos cenários de restrição de recursos computacionais (YANG et al., 2023). Ao ajustar o valor de ε, o método controla a quantidade média de FLOPs utilizados na inferência, permitindo que o modelo se adapte às condições do ambiente-alvo sem exigir retreinamento ou redesenho estrutural do modelo de detecção.

cabeças de detecção são otimizadas simultaneamente em um treinamento conjunto, utilizando uma função de perda ponderada, garantindo consistência em todas as saídas.

Resultados experimentais no conjunto de dados MS COCO demonstraram que o AdaDet atinge desempenho de estado da arte tanto em cenários de predição "a qualquer momento" *(anytime prediction)* quanto em cenários de inferência adaptativa. O sistema supera detectores convencionais, alcançando acurácia semelhante com significativamente menos operações de ponto flutuante por segundo (FLOPs). Além disso, a adaptabilidade do AdaDet a diferentes restrições computacionais sem a necessidade de retreinamento representa um avanço substancial, tornando-o adequado para implantação em ambientes com recursos limitados, como dispositivos de borda e dispositivos móveis.

O estudo não inclui nenhuma análise da dificuldade do input como parte de seu framework adaptativo. O método atual aplica uma política baseada apenas nas predições geradas pela rede, sem avaliar a complexidade intrínseca da entrada. A incorporação de uma análise da dificuldade do input poderia aprimorar ainda mais a eficácia do método, alinhando a alocação de recursos computacionais de forma mais estreita às demandas específicas de cada imagem.

4 CONJUNTO DE DADOS E PRÉ-PROCESSAMENTOS

A base de dados KITTI (GEIGER; LENZ; URTASUN, 2012) é amplamente reconhecida como uma referência em pesquisas de visão computacional, especialmente para tarefas de detecção de objetos em cenários urbanos. Dentro de seus diversos módulos, destaca-se o subconjunto voltado para a detecção de objetos 2D, que fornece anotações precisas em caixas delimitadoras (*bounding boxes*) para objetos em cenas reais de tráfego urbano. Esse subconjunto é composto por mais de 15.000 imagens anotadas, totalizando mais de 80.000 objetos rotulados, distribuídos em categorias como carros, pedestres e ciclistas. Cada anotação inclui informações detalhadas sobre a posição, tamanho e estado dos objetos, como oclusões parciais ou truncamentos.

As imagens da base KITTI foram capturadas utilizando câmeras estéreo de alta resolução, com dimensões de 1242x375 *pixels*, montadas em um veículo em movimento por áreas urbanas da cidade de Karlsruhe, Alemanha. O processo de aquisição incluiu condições variadas de iluminação, mudanças de perspectiva e diferentes níveis de densidade de tráfego, abrangendo desde vias congestionadas até cenários mais simples e abertos. Essa diversidade de condições visuais representa os desafios encontrados em aplicações reais de navegação autônoma, tornando a base uma escolha robusta e desafiadora para avaliação de algoritmos de detecção de objetos.

Figura 36 – Exemplo de imagem da base de dados KITTI (KRAUSS, 2022).

O subconjunto de detecção de objetos 2D mostra-se particularmente relevante para avaliar o sistema proposto, pois abrange cenários de diferentes níveis de complexidade visual. Essa variedade permite verificar em que medida o ajuste dinâmico do esforço computacional contribui para manter o desempenho do modelo em situações que vão desde cenas simples (poucos objetos e sem oclusões) até aquelas mais complexas (com objetos parcialmente ocluídos ou

em posições descentralizadas). Essa abordagem possibilita testar, de forma abrangente, a eficácia do sistema em contextos onde a eficiência e a precisão são igualmente cruciais, como em aplicações de navegação autônoma.

4.1 DESCRIÇÃO FORMAL DA BASE DE DADOS KITTI

A base de dados KITTI para detecção de objetos 2D é composta por um conjunto de imagens anotadas e suas respectivas informações de detecção. Formalmente, a base pode ser definida como:

$$\mathcal{D} = \{(\mathbf{X}_i, \mathcal{Y}_i)\}_{i=1}^{N}$$

Onde:

- $\mathcal{D}$ representa o conjunto de dados completo;
- N é o número total de amostras na base ($N = 15.000$);
- $\mathbf{X}_i \in \mathbb{R}^{H \times W \times C}$ é a i-ésima imagem da base, com dimensões $H = 375$, $W = 1242$ e $C = 3$ (representando as três componentes de cor: vermelho, verde e azul);
- $\mathcal{Y}_i = \{(c_j, \mathbf{b}_j)\}_{j=1}^{M_i}$ é o conjunto de anotações associadas à i-ésima imagem, contendo informações sobre M_i objetos rotulados.

Cada anotação $(c_j, \mathbf{b}_j)$ contém:

- $c_j \in \mathcal{C}$ é a classe do j-ésimo objeto, onde:

  $$\mathcal{C} = \{\text{Carro}, \text{Ciclista}, \text{Diversos}, \text{Pedestre}, \text{Pessoa\_sentada}, \text{Bonde}, \text{Caminhão}, \text{Van}\};$$

- $\mathbf{b}_j \in \mathbb{R}^4$ é a caixa delimitadora do j-ésimo objeto, representada como $\mathbf{b}_j = (x, y, w, h)$, onde:
  - x, y representam as coordenadas do canto superior esquerdo da caixa;
  - w, h correspondem à largura e altura da caixa, respectivamente.

Assim, uma amostra individual da base pode ser expressa como:

$$(\mathbf{X}_i, \mathcal{Y}_i) = \left(\mathbf{X}_i, \{(c_j, \mathbf{b}_j)\}_{j=1}^{M_i}\right)$$

4.2 PRÉ-PROCESSAMENTO E CÁLCULO DO ESCORE DE DIFICULDADE

A metodologia proposta por Ionescu (IONESCU et al., 2016) é amplamente reconhecida por sua inovação ao estimar a dificuldade de imagens em tarefas de visão computacional. O trabalho original introduz um método que utiliza anotações humanas e aprendizado profundo para calcular um escore de dificuldade, com base no tempo necessário para localizar alvos em uma imagem. Essa estimativa considera propriedades interpretáveis, como a densidade de objetos, a complexidade do plano de fundo e a similaridade entre distratores, características que impactam diretamente o desempenho de modelos de detecção.

Essa abordagem é particularmente relevante para este trabalho, pois fornece uma base sólida para quantificar a dificuldade de amostras em um conjunto de dados, um componente crítico no treinamento de estimadores dinâmicos. No entanto, para alinhá-la aos objetivos deste projeto, algumas adaptações foram necessárias. Essas modificações visam abordar limitações específicas da metodologia original, como a dependência de anotações humanas e a ausência de generalização para conjuntos de dados com propriedades visuais distintas.

O principal objetivo das adaptações realizadas é calcular um escore de dificuldade para cada amostra da base de dados, de forma automatizada e escalável, eliminando a necessidade de intervenção manual. Este escore servirá como uma métrica fundamental no treinamento de um modelo estimador capaz de prever a dificuldade das imagens em tempo real, integrando-se diretamente à arquitetura dinâmica proposta neste trabalho.

As modificações introduzidas incluem a reformulação das propriedades analisadas para enfatizar métricas baseadas em redes neurais convolucionais (CNNs) pré-treinadas, que fornecem representações mais robustas e generalizáveis das características da imagem. Além disso, foram excluídas características altamente subjetivas, como a percepção de dificuldade por humanos, substituindo-as por métricas derivadas diretamente do desempenho do modelo. Essa abordagem não apenas reduz a subjetividade, mas também torna o processo aplicável a bases de dados heterogêneas, sem a necessidade de reanotações específicas.

4.3 CÁLCULO DO ESCORE DE DIFICULDADE

O escore S representa a dificuldade relativa da amostra em relação ao restante do conjunto de dados. Essa abordagem permite a criação de um ranking sistemático das imagens, útil para ordenar as amostras de acordo com a complexidade percebida por um modelo. Para calcular

S, baseamos nossa metodologia em métricas objetivas derivadas tanto das características intrínsecas da imagem quanto de informações presentes no *groundtruth*, como o desempenho de modelos pré-treinados.

Para garantir que cada métrica contribua proporcionalmente para o cálculo do escore final, todas as métricas são normalizadas para o intervalo $[0, 1]$. A normalização de cada métrica f_i é realizada como:

$$f_i = \frac{\text{metric}_i - \text{min}_i}{\text{max}_i - \text{min}_i}$$

onde metric_i é o valor bruto da métrica para a imagem atual, e min_i e max_i são os valores mínimo e máximo de metric_i no conjunto de dados.

As métricas utilizadas no cálculo de S são as seguintes:

1. **Número de Objetos (f_1)**: Representa o número total de objetos individuais dentro da imagem. Um maior número de objetos pode aumentar a complexidade visual da cena, dificultando o processo de detecção. A métrica é calculada como:

$$f_1 = \frac{M - \text{min}_M}{\text{max}_M - \text{min}_M}$$

2. **Área Média dos Objetos (f_2)**: Mede a média da área ocupada pelos objetos na imagem, expressa como uma porcentagem da área total da imagem. Objetos maiores são geralmente mais fáceis de detectar, enquanto objetos menores aumentam a dificuldade. A métrica é calculada como:

$$f_2 = \frac{1}{M} \sum_{j=1}^{M} \frac{w_j \cdot h_j}{H \cdot W}$$

3. **Não Centralização (f_3)**: Avalia a distância média dos objetos ao centro da imagem. Objetos distantes do centro tendem a ser mais difíceis de detectar, pois geralmente estão fora da área de foco principal. A métrica é definida como:

$$f_3 = \frac{1}{M} \sum_{j=1}^{M} \sqrt{\left(\frac{x_j - W/2}{W}\right)^2 + \left(\frac{y_j - H/2}{H}\right)^2}$$

4. **Número de Classes Distintas (f_4)**: Refere-se à diversidade de classes presentes na imagem, medindo a quantidade de categorias distintas de objetos. Uma maior diversidade pode aumentar a complexidade devido à variação nas formas e aparências. A métrica é calculada como:

$$f_4 = \frac{\text{num\_classes} - \text{min}_{\text{classes}}}{\text{max}_{\text{classes}} - \text{min}_{\text{classes}}}$$

5. **Número de Objetos Ocluídos (f_5)**: Mede a proporção de objetos que estão parcialmente ocultos por outros objetos na imagem. Objetos ocluídos são mais difíceis de detectar, pois partes importantes podem estar escondidas. A métrica é dada por:

$$f_5 = \frac{\text{num\_occluded\_objects}}{M}$$

6. **Média de Confiança (f_6)**: Calcula a média dos escores de confiança atribuídos por um comitê de redes neurais convolucional (CNN) pré-treinadas na ImageNet (Resnet50, Vgg16, Mobilenetv2, Efficientnetb0) a todos os objetos na imagem. Escores mais baixos indicam imagens mais difíceis, enquanto escores mais altos indicam maior facilidade para o modelo classificar. A métrica é calculada como:

$$f_6 = \frac{1}{M} \sum_{j=1}^{M} c_j$$

7. **Escore de Concordância (f_7)**: Mede o nível de concordância entre múltiplos modelos ao classificar os objetos na imagem. É definido como a fração de modelos de um comitê de redes neurais convolucional (CNN) pré-treinadas ImageNet (Resnet50, Vgg16, Mobilenetv2, Efficientnetb0) que preveem a classe majoritária para cada objeto. Um escore mais alto reflete maior consenso entre os modelos, enquanto um escore mais baixo indica maior divergência. A métrica é calculada como:

$$f_7 = \frac{1}{M} \sum_{j=1}^{M} \frac{N_{\text{majority},j}}{N}$$

   onde $N_{\text{majority},j}$ é o número de modelos que preveem a classe majoritária do objeto j, e N é o número total de modelos avaliados.

O escore final S é obtido pela soma ponderada das métricas normalizadas:

$$S = 0.15 \cdot f_1 + 0.15 \cdot (1 - f_2) + 0.10 \cdot f_3 + 0.10 \cdot f_4 + 0.10 \cdot f_5 + 0.15 \cdot f_6 + 0.25 \cdot f_7$$

Por fim, a base de dados completa pode ser descrita como um conjunto $\mathcal{D}$, onde:

$$\mathcal{D} = \{(\mathbf{X}_i, \mathcal{Y}_i, S_i)\}_{i=1}^{N}$$

Impacto da Normalização

A normalização garante que cada métrica contribua de forma proporcional para o cálculo do escore S, independentemente de suas escalas originais. Além disso, o escore S reflete a dificuldade relativa de cada amostra no conjunto de dados, permitindo comparações consistentes entre imagens e a criação de estratégias adaptativas para detecção de objetos.

Esquema de Pesos

- **Pesos de 0.15**: As características com peso 0.15 são consideradas altamente influentes no cálculo do escore de dificuldade. Entre elas estão o número de objetos, a área média inversa ocupada pelos objetos, o escore de consistência e o escore de concordância. Essas métricas têm impacto significativo na complexidade da tarefa de detecção de objetos.
- **Pesos de 0.10**: Características com peso 0.10 possuem um impacto moderado no cálculo do escore de dificuldade. Isso inclui a não centralização, o número de classes, o número de objetos ocluídos e a área média de interseção. Embora relevantes, essas métricas são ligeiramente menos críticas que aquelas com pesos mais altos.

Racionalidade - Esquema de Pesos e Normalização

Os pesos foram determinados empiricamente, com base em análise visual dos níveis de dificuldades. O objetivo principal foi equilibrar a influência de cada métrica, refletindo com precisão sua contribuição relativa para a dificuldade de detecção. A normalização, por sua vez, garante que cada métrica contribua proporcionalmente, evitando que métricas com amplitudes numéricas maiores dominem o cálculo do escore.

Ao combinar essas métricas ponderadas e normalizadas, obtemos um único escore de dificuldade para cada imagem. Esse escore representa quantitativamente os desafios que um modelo de detecção pode enfrentar ao processar a imagem. Ele permite o ranqueamento das amostras com base em sua dificuldade, auxiliando na adaptação de estratégias de detecção e, potencialmente, aprimorando a robustez e a precisão do modelo em ambientes visuais complexos.

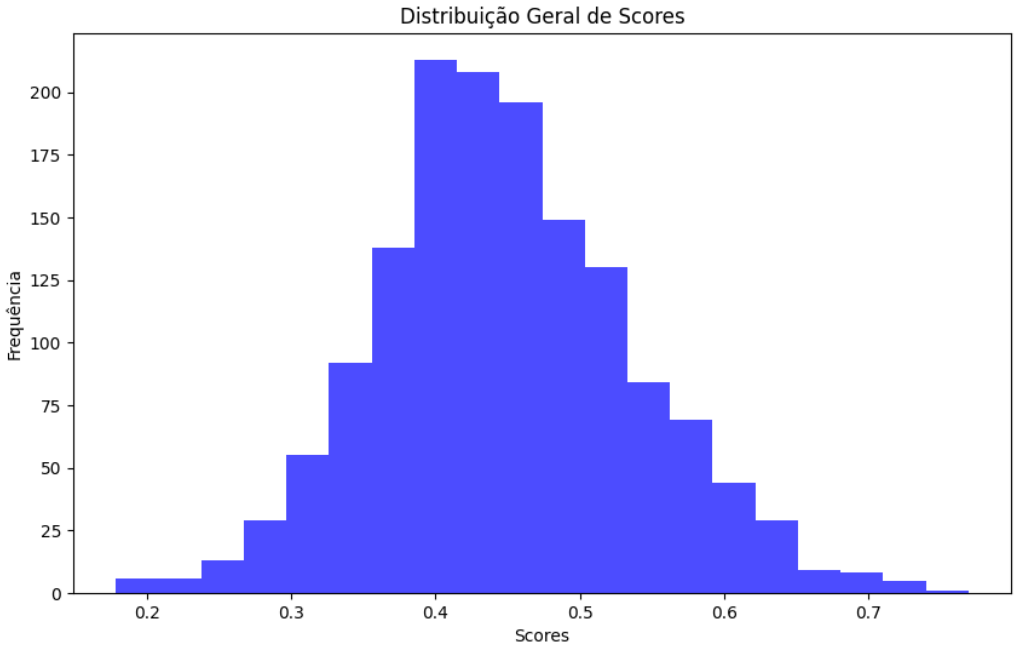


Figura 37 – Distribuição geral dos scores calculados na base KITTI. Figura elaborada pelo autor.

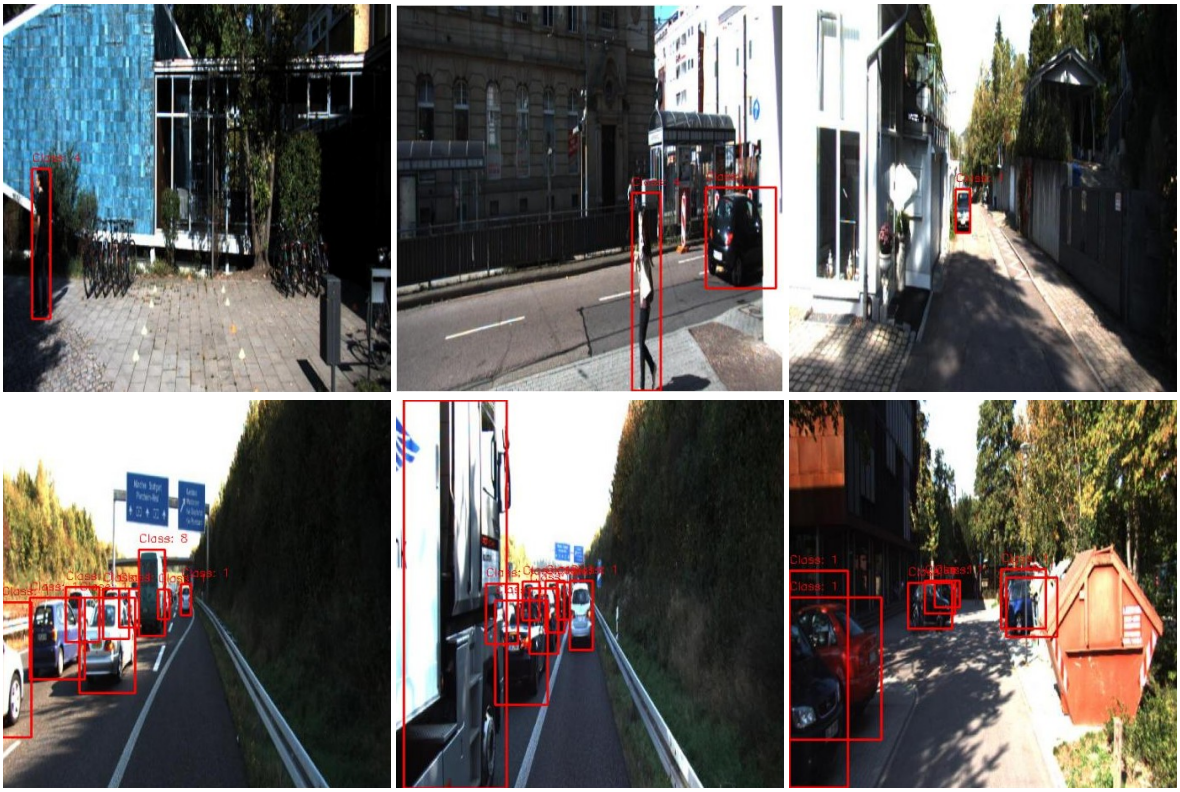


Figura 38 – Comparação entre exemplos fáceis (linha superior) e difíceis (linha inferior) com base nos scores de dificuldade. As imagens mostram diferentes níveis de complexidade em termos de objetos detectados, condições de iluminação e sobreposições. Figura elaborada pelo autor.

5 METODOLOGIA E DESCRIÇÃO DA ARQUITETURA PROPOSTA

5.1 INTRODUÇÃO

A implementação de uma arquitetura dinâmica para detecção de objetos requer uma abordagem estruturada que integre múltiplos componentes de maneira eficiente. Este capítulo apresenta a metodologia adotada neste trabalho, abrangendo desde uma visão geral do sistema até os detalhes específicos de seus principais componentes e das estratégias de treinamento e inferência.

Inicialmente, será descrita a metodologia de pré-processamento dos dados, além da apresentação da base de dados utilizada para os experimentos. Depois, será fornecida uma visão geral do sistema proposto, destacando como seus componentes se conectam e colaboram para alcançar uma detecção eficiente e adaptativa. Essa seção oferecerá uma perspectiva geral sobre o fluxo de processamento e a interação entre os módulos principais.

Em seguida, será introduzido o Estimador de Dificuldades, que desempenha um papel central na política adaptativa do sistema. Serão apresentados a proposta de uma métrica para quantificar a dificuldade das imagens, o método utilizado para rotular a base de dados com base nessa métrica e o modelo desenvolvido para estimar a dificuldade em tempo de execução. A seção também incluirá detalhes sobre o treinamento do estimador, como a configuração dos hiperparâmetros e as métricas de avaliação.

O capítulo prossegue com a descrição do *Early Exit Detector*, que engloba as adaptações realizadas no *backbone*, ResNet101, e na arquitetura SSD para suportar saídas antecipadas. Serão discutidas as modificações estruturais necessárias, incluindo a posição das saídas intermediárias e a definição das ramificações (*branches*). Detalhes sobre o treinamento da arquitetura, bem como o funcionamento da inferência, serão abordados nesta seção. A descrição é genérica, no sentido de que serve para um número arbitrário de saídas, porém, neste trabalho, foi implementada uma únia saída antecipada.

Posteriormente, será apresentada a Política de Saída, responsável por determinar dinamicamente o ponto de término da inferência com base na dificuldade da entrada. A descrição formal da política será acompanhada por uma análise crítica de suas decisões e uma discussão sobre possibilidades futuras, como o uso de aprendizado para otimização dos limiares.

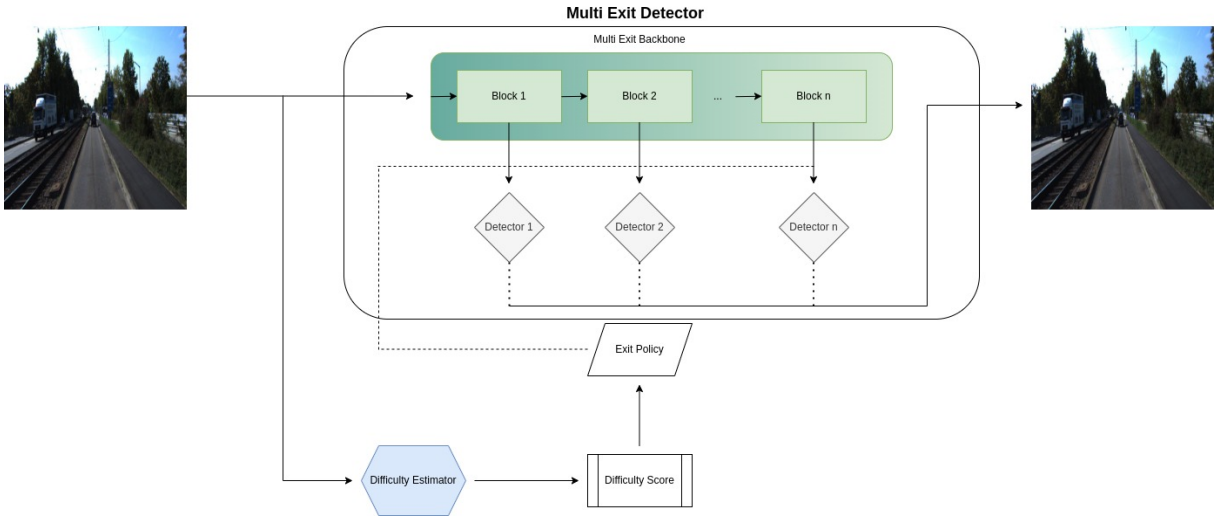


Figura 39 – Visão Geral do Sistema com um número arbitrário de saídas, mostrando a conexão entre os módulos. Figura elaborada pelo autor.

5.2 DESCRIÇÃO DO SISTEMA PROPOSTO

5.2.1 Visão Geral do Sistema

O sistema proposto introduz uma abordagem inovadora para a detecção eficiente de objetos, composta por dois componentes principais: um avaliador de complexidade de imagens e uma rede de detecção de objetos com múltiplas saídas antecipadas (multi-exit). Essa arquitetura foi desenvolvida para enfrentar os desafios de eficiência computacional e adaptabilidade em cenários dinâmicos, combinando análise de complexidade com inferência adaptativa.

O principal diferencial do sistema reside na integração eficiente entre esses dois componentes. Ao contrário das abordagens convencionais, onde a inferência é conduzida de forma estática e uniforme para todas as entradas, o sistema avalia dinamicamente a complexidade de cada imagem, determinando o ponto ideal para encerrar o processamento. Isso é alcançado por meio da inclusão de saídas antecipadas na rede de detecção, o que possibilita o processamento rápido de entradas mais simples e o aproveitamento pleno dos recursos computacionais para entradas mais complexas. Essa estratégia otimiza o uso de recursos ao mesmo tempo que mantém a precisão nas detecções.

Durante a inferência, a imagem de entrada é inicialmente processada pelo Estimador de Dificuldade, que avalia sua complexidade com base em características visuais e retorna um escore no intervalo [0, 1]. Um escore próximo de 0 indica uma imagem considerada "fácil"para detecção, enquanto valores próximos a 1 indicam maior complexidade. Esse escore serve como

parâmetro para uma política de saída predefinida, responsável por determinar a saída apropriada da rede de detecção para a entrada em questão.

O Estimador de Dificuldade foi projetado para ser rápido e compacto, minimizando seu impacto no tempo total de inferência. Essa eficiência é essencial para garantir um dos principais benefícios do sistema: a redução do tempo de processamento para imagens mais simples, sem comprometer a precisão necessária para entradas mais desafiadoras.

A política de saída desempenha um papel crucial ao regular qual saída será utilizada para finalizar o processamento de uma imagem. Neste trabalho, a política adota uma abordagem baseada em limiares (*thresholds*) equidistantes que dividem o intervalo de complexidade [0, 1] em regiões de acordo com o número de saídas do detector. Para um detector com n saídas antecipadas, os limiares definem as transições entre essas regiões, determinando qual saída processará cada imagem. Entradas mais simples são processadas por saídas iniciais, enquanto aquelas mais complexas avançam para camadas mais profundas.

Por exemplo, no caso de duas saídas, existem apenas duas regiões, separadas por um limiar $t_1 = 0.5$. Imagens com escore abaixo de 0.5 são processadas pela primeira saída, enquanto imagens com escore igual ou superior a 0.5 são processadas pela segunda saída. Esse mecanismo categoriza as imagens em grupos de dificuldade e direciona os recursos computacionais de forma apropriada a cada grupo, otimizando o desempenho geral do sistema.

A arquitetura de detecção multi-saídas é implementada como uma versão customizada de uma rede de detecção tradicional, que neste trabalho se baseia no Single Shot MultiBox Detector (SSD). Na versão convencional, a arquitetura possui um único caminho entre a entrada e a saída. Na versão customizada, são introduzidas saídas antecipadas em camadas específicas do backbone. Essas saídas consistem em conexões diretas entre as features extraídas nessas camadas e detectores dedicados a cada estágio. Com isso, o modelo pode realizar detecção em diferentes pontos da rede, dependendo da decisão política que regula o ponto de término da execução.

Cada caminho computacional entre a entrada e uma saída, denominado como ramo, é definido pela execução do backbone até o ponto especificado pela política de saída, seguido pelo processamento realizado pelo detector associado a essa saída. Essa estrutura modular permite ao sistema adaptar-se dinamicamente às características da entrada, maximizando a eficiência computacional sem comprometer a precisão da detecção.

5.2.2 Estimador de Dificuldade

O estimador de dificuldade é uma peça central no sistema proposto, responsável por prever o escore $\mathbf{S}$ associado a cada imagem $\mathbf{X}$. Essa predição orienta as decisões dinâmicas do modelo durante a inferência, permitindo otimizar o balanceamento entre precisão e custo computacional. Para alcançar esses objetivos, foi selecionada a versão mais leve da arquitetura EfficientNet, reconhecida por sua eficiência em termos de uso de recursos e desempenho, mesmo em cenários computacionalmente restritos.

A arquitetura base utilizada neste trabalho é a EfficientNet-Lite0, projetada originalmente para tarefas de classificação em dispositivos com recursos computacionais limitados. Para adaptar esta arquitetura à tarefa de regressão, onde o objetivo é prever um valor escalar representando o *score* de dificuldade de uma imagem no intervalo [0, 1], foram realizadas as seguintes modificações na estrutura original.

Inicialmente, a camada de saída da EfficientNet-Lite0, projetada para classificação multi-classe, foi substituída por uma nova cabeça para regressão. Na configuração original, a saída é composta por uma camada densamente conectada (*fully connected, FC*) com C neurônios, onde C representa o número de classes. Essa camada utiliza a ativação *softmax* para gerar uma distribuição de probabilidades sobre as classes. Para a tarefa de regressão, essa camada foi reprojetada para mapear as características extraídas pelo *backbone* em um único valor escalar.

Primeiramente, foi mantida a etapa de *global average pooling* (GAP), que reduz as dimensões espaciais ($H \times W$) do tensor de características extraído pela última camada convolucional para um vetor unidimensional de tamanho $D = 1280$, onde D representa o número de canais na saída da EfficientNet-Lite0. Essa operação é definida como:

$$z_d = \frac{1}{H \cdot W} \sum_{i=1}^{H} \sum_{j=1}^{W} x_{i,j,d}, \quad d \in \{1, \ldots, D\},$$

onde $x_{i,j,d}$ é o valor do elemento na posição (i, j, d) do tensor de entrada para o canal d, e z_d é o valor do d-ésimo elemento do vetor resultante $z \in \mathbb{R}^D$.

Após o *global pooling*, o vetor resultante passa por uma camada densamente conectada com 1 neurônio, cuja função é mapear z para um escalar s. Esta camada é definida como:

$$s = \sigma\left(w^\top z + b\right),$$

onde $w \in \mathbb{R}^D$ é o vetor de pesos da camada, $b \in \mathbb{R}$ é o viés, e σ representa a função de

ativação *sigmoid*, dada por:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

Essa função garante que o *score* de dificuldade s esteja no intervalo $[0, 1]$, como requerido pela tarefa. A escolha da ativação *sigmoid* reflete a necessidade de produzir uma saída normalizada e contínua.

A adaptação também utilizou os pesos pré-treinados da EfficientNet-Lite0 na ImageNet como inicialização para as camadas convolucionais, explorando o *transfer learning*. Inicialmente, as camadas convolucionais foram congeladas para treinar exclusivamente a nova cabeça de regressão. Posteriormente, foi realizado o *fine-tuning* das camadas mais profundas para ajustar os pesos da rede às características do dataset KITTI.

Por fim, o modelo modificado mantém a eficiência computacional característica da EfficientNet-Lite0, com aproximadamente 4,4 milhões de parâmetros, mas agora adaptado para a tarefa de prever *scores* contínuos, com as modificações matemáticas e arquiteturais necessárias para atender aos requisitos de eficiência da aplicação proposta.

Tabela 7 – Resumo da EfficientNet-Lite0 Adaptada

Estágio	Operador	Configuração	Stride	Saída
Input	Conv 3×3	$3 \to 32$, BN, ReLU6	2	$112 \times 112 \times 32$
MBConv1	*MBConv*	$32 \to 16$, 3×3	1	$112 \times 112 \times 16$
MBConv6	*MBConv*	$16 \to 24$, 3×3	2 (1º bloco)	$56 \times 56 \times 24$
MBConv6	*MBConv*	$24 \to 40$, 5×5	2 (1º bloco)	$28 \times 28 \times 40$
MBConv6	*MBConv*	$40 \to 80$, 3×3	2 (1º bloco)	$14 \times 14 \times 80$
MBConv6	*MBConv*	$80 \to 112$, 5×5	1	$14 \times 14 \times 112$
MBConv6	*MBConv*	$112 \to 192$, 5×5	2 (1º bloco)	$7 \times 7 \times 192$
Conv Final	Conv 1×1	$192 \to 1280$, BN, ReLU6	1	$7 \times 7 \times 1280$
Adaptada	Global Pool + Dropout	Vetor 1×1280	-	1×1280
Saída	FC + Sigmoid	$1280 \to 1$	-	Score de Dificuldade (0 a 1)

5.2.3 Política de saída

A política proposta é formalmente definida como segue:

Dado um valor $d \in [0, 1]$, retornado pelo Estimador de Dificuldade, e um conjunto de n saídas $S = \{s_1, s_2, \ldots, s_n\}$, o intervalo $[0, 1]$ é dividido em n regiões por $n - 1$ limiares t_i,

onde cada limiar é definido como:

$$t_i = \frac{i}{n}, \quad \text{para } i = 1, 2, \ldots, n-1.$$

A saída $s_k \in S$ é selecionada de acordo com a seguinte regra:

$$s_k \text{ é escolhida se } t_{k-1} \leq d < t_k,$$

onde $t_0 = 0$ e $t_n = 1$.

Assim, para cada d, a inferência é direcionada para a saída correspondente s_k, com base na dificuldade estimada da entrada e nos limiares predefinidos.

5.2.4 Metodologia do Detector Multi-Exit

A arquitetura base escolhida para o desenvolvimento do detector multi-exit é o *Single Shot MultiBox Detector (SSD)*, devido à sua simplicidade estrutural e eficiência. Sua arquitetura é composta por um *backbone* claramente delimitado, neste projeto uma ResNet101, que extrai as características das imagens, e uma série de camadas adicionais destinadas à extração de features em um número maior de escalas. Um aspecto distintivo do SSD é a ausência de interconexões complexas ou retroalimentações, características que tornam a arquitetura modular e relativamente fácil de customizar.

5.2.5 Estrutura da Arquitetura

A arquitetura de base adotada neste trabalho é o SSD, ao qual foi incorporada a ResNet101 como *backbone* para extração de características. A escolha dessa rede se justifica pela necessidade de mitigar as limitações observadas no SSD original, que empregava a VGG16 como *backbone* e demonstrou desempenho insatisfatório em experimentos preliminares. Esse baixo desempenho dificultou tanto a comparação entre o modelo de referência e o sistema proposto quanto a avaliação das melhorias introduzidas.

Por sua vez, a ResNet101, graças à maior profundidade e ao potencial de aprender representações mais discriminativas, fornece *feature maps* de melhor qualidade.

Para a tarefa de detecção, a ResNet101 preserva saídas intermediárias de camadas específicas para gerar *feature maps* em múltiplas resoluções. As camadas selecionadas para retorno são:

- **'layer1.2.conv3'**: Saída da última convolução do primeiro bloco residual, gerando um *feature map* com $C_1 = 256$ canais.
- **'layer2.3.conv3'**: Saída da última convolução do segundo bloco residual, com $C_2 = 512$ canais.
- **'layer3.22.conv3'**: Saída da última convolução do terceiro bloco residual, com $C_3 = 1024$ canais.
- **'layer4.2.conv3'**: Saída da última convolução do quarto bloco residual, com $C_4 = 2048$ canais.

Os *feature maps* extraídos dessas camadas possuem resoluções decrescentes devido ao uso de convoluções com *stride* e operações de *pooling* ao longo da rede, permitindo que diferentes camadas capturem características em escalas distintas.

Para complementar os *feature maps* da ResNet101, o modelo adiciona três camadas convolucionais extras. Essas camadas são projetadas para:

- Reduzir progressivamente a resolução espacial dos *feature maps*.
- Adaptar o número de canais de saída para níveis adequados de abstração em cada escala.

Cada camada adicional é composta por duas operações:

1. **Convolução 1×1:** Reduz a dimensionalidade dos canais de entrada (C_{in}) para uma dimensão intermediária (C_{mid}). Formalmente:

$$f_{1\times1}(x) = \text{ReLU}(\text{Conv}_{1\times1}(x, W_{1\times1})),$$

onde $W_{1\times1} \in \mathbb{R}^{C_{\text{mid}} \times C_{\text{in}} \times 1 \times 1}$ são os pesos do filtro.

2. **Convolução 3×3 com *stride* 2:** Reduz a resolução espacial do *feature map* ($H \times W \to H/2 \times W/2$). Formalmente:

$$f_{3\times3}(x) = \text{ReLU}(\text{Conv}_{3\times3}(x, W_{3\times3})),$$

onde $W_{3\times3} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{mid}} \times 3 \times 3}$ são os pesos do filtro.

Essas operações são combinadas para formar três camadas sequenciais, cada uma gerando um novo *feature map*. A tabela abaixo resume os parâmetros dessas camadas adicionais:

Tabela 9 – Parâmetros das Camadas Adicionais

Camada	C_{in}	C_{mid}	C_{out}
Extra1	2048	512	512
Extra2	512	256	256
Extra3	256	256	256

A saída total do *backbone* e das camadas extras é um conjunto de sete *feature maps*, com resoluções decrescentes. Esses mapas são utilizados para detectar objetos em diferentes escalas. A ordem das resoluções é definida como:

$$\{H_1 \times W_1, H_2 \times W_2, \ldots, H_7 \times W_7\},$$

onde $H_{i+1} = H_i/2$ e $W_{i+1} = W_i/2$. O número de canais associados a cada mapa é dado por:

$$\{C_1, C_2, C_3, C_4, C_5, C_6, C_7\} = \{256, 512, 1024, 2048, 512, 256, 256\}.$$

Cada *feature map* é associado a um conjunto de *default boxes* com diferentes proporções de aspecto ($\{1.0, 2.0, 0.5, 3.0\}$) e escalas específicas ($[0.05, 0.15, 0.3, 0.45, 0.6, 0.75, 0.85, 1.0]$). O número total de caixas geradas por mapa é proporcional ao tamanho do mapa e ao número de proporções de aspecto.

5.2.6 Fluxo Geral

1. A entrada é processada pela ResNet101, gerando quatro *feature maps*.
2. Esses mapas são suplementados pelas camadas adicionais, formando um conjunto de sete mapas.
3. Cada *feature map* é utilizado para prever classes e localizações para os *default boxes* associados.

Essa arquitetura combina profundidade, eficiência computacional e suporte a detecção em multiescalas, tornando-a uma base robusta para o desenvolvimento do modelo modificado.

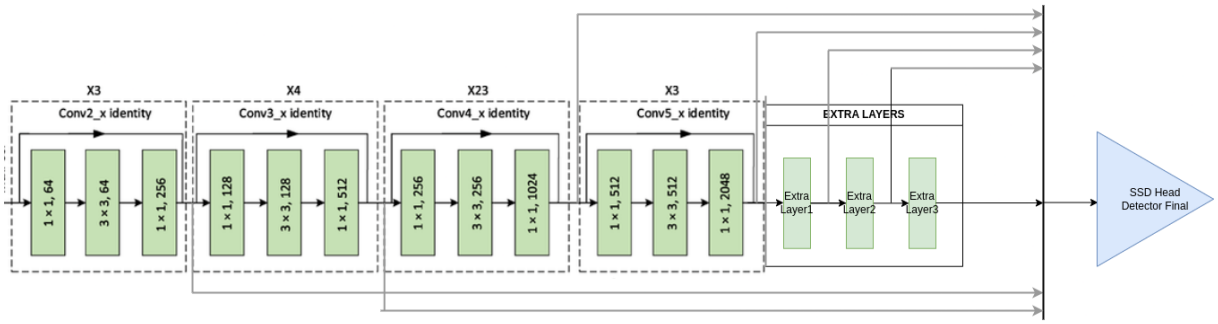


Figura 40 – Ilustração do SSD implementado com a Resnet101 como *backbone*, o modelo base. Figura elaborada pelo autor.

5.2.7 Inserção de Ramos no Modelo e Principais Modificações

Neste trabalho, as modificações realizadas na arquitetura original serão apresentadas em dois níveis de detalhamento. Primeiramente, será fornecida uma descrição genérica das alterações, com o objetivo de introduzir os conceitos fundamentais e o racional por trás das mudanças propostas. Posteriormente, serão descritos os detalhes específicos e as particularidades da versão da arquitetura testada neste projeto.

A principal modificação realizada na arquitetura do SSD é a introdução de **ramos de saída antecipadas** em pontos arbitrários ao longo do *backbone*. Esses ramos são projetados para fornecer saídas intermediárias, possibilitando que a inferência seja encerrada antecipadamente com base na dificuldade da entrada. Cada ramo consiste em uma conexão dedicada entre as saídas das camadas anteriores e um detector específico associado a essa saída. O caminho entre a camada de entrada e a última saída é idêntico ao modelo base.

Adaptação das *Features* e Estrutura dos Ramos: Para garantir a compatibilidade entre as saídas do *backbone* e os detectores intermediários, cada ramo é estruturado como uma sequência de camadas convolucionais. Essas convoluções têm a função de refinar as características extraídas das camadas anteriores, ajustando tanto a dimensionalidade dos canais quanto a resolução espacial dos *feature maps*. Após essa sequência de convoluções, as características refinadas são processadas pelas camadas extras previamente descritas, responsáveis por produzir os *feature maps* adicionais com maior abstração e suporte a detecções em múltiplas escalas.

Detectores Dedicados: Cada ramo inclui um detector completo e independente, projetado para realizar a detecção de objetos com base nas características extraídas e refinadas ao longo do ramo. A estrutura desses detectores é idêntica à do detector principal, garantindo

consistência entre as saídas intermediárias e finais. Os parâmetros, como ncoras, escalas e proporções de aspecto, são configurados uniformemente em todos os detectores, assegurando que as predições sejam interpretadas de forma consistente em toda a arquitetura.

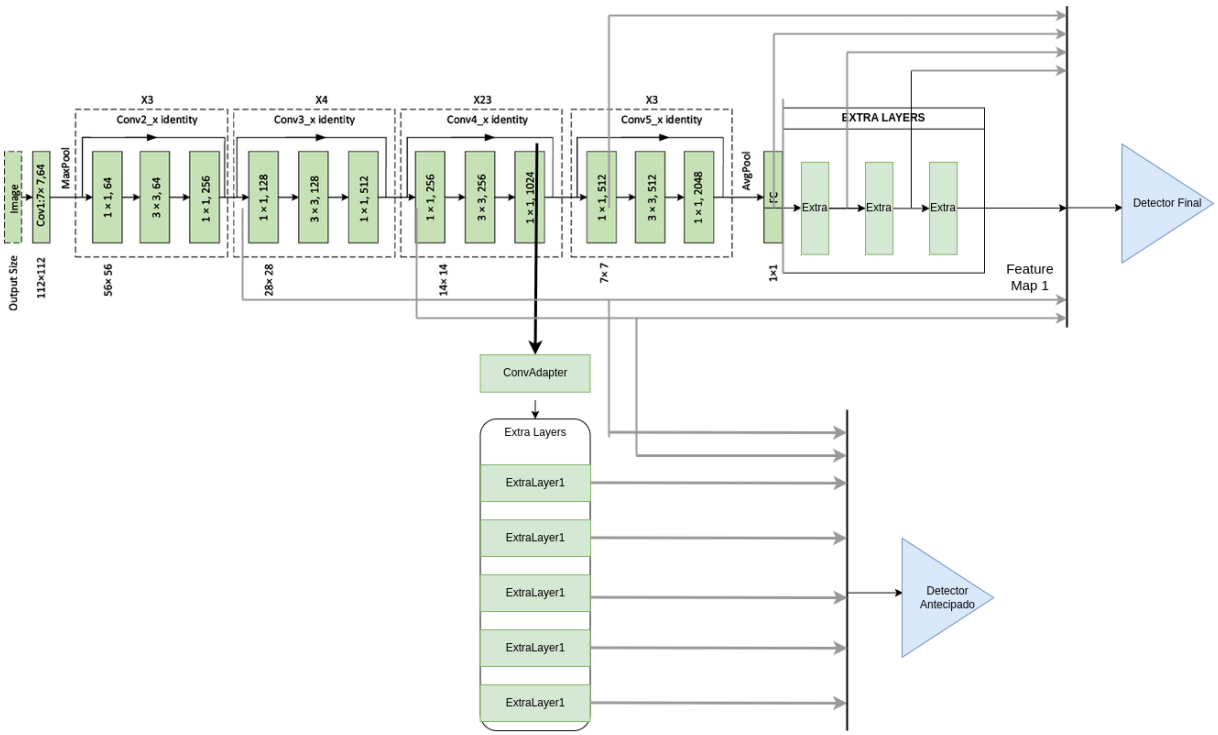


Figura 41 – Ilustração do SSD modificado com uma saída antecipada. Figura elaborada pelo autor.

O *backbone* da arquitetura é estruturado como uma sequência de blocos residuais. indexados, onde cada bloco, denotado por B_n para $n \in \{0, 1, \ldots, m\}$. No caso da Resnet101, $m = 33$

$$B_n(x) = x + F(x, \Theta_n),$$

onde:

- x é a entrada do bloco.
- $F(x, \Theta_n)$ é a transformação aprendida pelo bloco, composta pelas convoluções e ativações ReLU, com Θ_n representando os pesos associados às convoluções do bloco.

Para a inserção de uma saída antecipada (*early exit*), escolhe-se um índice n tal que $0 < n < m$, onde n determina o último bloco residual a ser executado antes da saída antecipada. Na arquitetura ResNet101, cada saída antecipada é posicionada após a conclusão

de um bloco residual completo, ou seja, imediatamente após a soma da conexão residual e a aplicação da última função de ativação ReLU do bloco B_n.

A estrutura de um ramo antecipado inserido após o bloco residual B_n consiste em uma sequência de operações definidas como segue:

Uma camada convolucional é aplicada diretamente sobre a saída do bloco residual, formalmente representada por:

$$\mathbf{h}_n = f_{\mathsf{conv}}(B_n(\mathbf{x}), \theta'_n) \tag{5.1}$$

onde $B_n(\mathbf{x}) = \mathbf{x} + F(\mathbf{x}, \Theta_n)$, com $F(\mathbf{x}, \Theta_n)$ representando a transformação aprendida pelo bloco residual. O número de canais de entrada da convolução é igual ao número de canais de saída do bloco residual anterior. Essa convolução tem o objetivo de refinar as representações extraídas e garantir compatibilidade dimensional para as operações subsequentes.

Depois, uma função de ativação ReLU é aplicada à saída da convolução, introduzindo não-linearidade e aprimorando a capacidade discriminativa das representações extraídas:

$$\mathbf{h}_n^{\mathsf{relu}} = \mathsf{ReLU}(\mathbf{h}_n^p) \tag{5.2}$$

O *feature map* resultante, $\mathbf{h}_n$, é utilizado diretamente pelo detector dedicado, mas não é a única fonte de informação para as predições. De maneira similar ao modelo base, esse *feature map* também é processado por uma sequência de camadas extras (*extra layers*) que geram novos *feature maps* em escalas menores. Esses *feature maps* adicionais complementam as informações extraídas, possibilitando que o detector dedicado realize predições de objetos em múltiplas resoluções e escalas.

Dessa forma, o ramo antecipado mantém a capacidade multiescala característica do *backbone* principal, assegurando que as predições sejam realizadas com base em representações refinadas e escaláveis. Essas *extra layers* adicionais são projetadas para compensar os *feature maps* que não foram gerados devido à execução parcial do *backbone*. Entretanto, como o número de *feature maps* disponíveis e seus formatos (*shapes*) variam dependendo do momento da interrupção, os parâmetros C_{in} e C_{out} das camadas convolucionais precisam ser ajustados para cada ponto de saída antecipada. Esses parâmetros garantem que as dimensões dos canais sejam compatíveis tanto com o *feature map* anterior quanto com as representações requeridas para o detector dedicado.

Por fim, um *head-detector* dedicado é inserido após a ativação ReLU. Esse detector é estruturalmente idêntico ao utilizado na saída final da rede e realiza a predição de objetos com base nas representações extraídas. Sua formulação pode ser representada como:

$$\hat{\mathbf{y}}_n = f_{\text{detector}}(\mathbf{h}_n^{\text{relu}}) \tag{5.3}$$

Os parâmetros deste detector, como *anchor generator*, escalas, proporções de aspecto e demais hiperparâmetros, são mantidos idênticos ao detector principal da saída final. Essa escolha preserva a coerência do modelo e assegura que as características extraídas sejam interpretadas de forma consistente ao longo dos diferentes ramos.

Neste trabalho, inserção do ramo antecipado foi realizada após o décimo bloco da terceira camada residual (`layer3`) do *backbone*. Nesse contexto, o índice n, que representa o posicionamento do ramo na sequência de blocos residuais da ResNet101, pode ser formalizado como:

$$n = n_{\text{layer1}} + n_{\text{layer2}} + 10,$$

onde:

- $n_{\text{layer1}} = 3$ é o número de blocos residuais em `layer1`;
- $n_{\text{layer2}} = 4$ é o número de blocos residuais em `layer2`;
- 10 é o índice do bloco na `layer3`, onde o ramo foi inserido.

Portanto, o valor de n varia em função do índice x e do total acumulado de blocos nas camadas anteriores, resultando em 17 camadas.

A escolha da quantidade de ramos e suas posições impacta diretamente a qualidade do sistema, porém a análise detalhada sobre a relação e a qualidade das predições nas saídas antecipadas não faz parte do escopo deste trabalho. O valor de n foi definido experimentalmente, buscando um equilíbrio entre granularidade e abstração, suficiente para demonstrar os benefícios do sistema proposto. Esse processo considerou tanto a eficácia do modelo quanto a viabilidade de manter a capacidade multiescala da arquitetura.

A estrutura do ramo antecipado é composta por uma sequência de operações projetadas para processar o *feature map* gerado pela saída B_n do *backbone*.

A saída $B_{17}(\mathbf{x})$, gerada pelo bloco residual, é processada por uma convolução inicial, que ajusta o número de canais do *feature map* para torná-lo compatível com as camadas extras subsequentes. Essa operação é definida como:

$$\mathbf{h}_n = f_{\text{conv}}(B_n(\mathbf{x}), \theta'_n),$$

onde:

- f_{conv} é uma convolução 1×1 (`nn.Conv2d`) com $C_{\text{in}} = \text{bn\_out}$ e $C_{\text{out}} = 2048$, configurada com `stride=1` e `padding=0`.
- $\mathbf{h}_n$ é o *feature map* ajustado para o formato de entrada requerido pelos *extra layers*.

Após o mapeamento inicial, $\mathbf{h}_n$ é processado por uma sequência de *extra layers*, que refinam as representações e geram novos *feature maps* em escalas progressivamente menores. A estrutura de cada camada extra é detalhada na Tabela 10.

Tabela 10 – Configuração das *Extra Layers*

Layer	Convolução	Canais de Entrada	Canais de Saída	Kernel Size	Stride
1	1×1	1024	1024	1×1	1
	3×3	1024	1024	3×3	2
2	1×1	1024	1024	1×1	1
	3×3	1024	2048	3×3	2
3	1×1	2048	1024	1×1	1
	3×3	1024	512	3×3	2
4	1×1	512	256	1×1	1
	3×3	256	256	3×3	2
5	1×1	256	256	1×1	1
	3×3	256	256	3×3	2

A saída dos *feature maps* gerados pelo *backbone* e pelas *extra layers* serve como entrada para o detector dedicado, totalizando sete escalas distintas. Esses *feature maps* apresentam resoluções decrescentes, estruturadas como:

$$\{H_1 \times W_1, H_2 \times W_2, \ldots, H_7 \times W_7\},$$

onde $H_{i+1} = H_i/2$ e $W_{i+1} = W_i/2$. O número de canais associados a cada *feature map* é dado por:

$$\{C_1, C_2, C_3, C_4, C_5, C_6, C_7\} = \{256, 512, 1024, 2048, 512, 256, 256\}.$$

O detector dedicado em cada escala é estruturalmente idêntico ao modelo base original. Para cada *feature map*, o detector aplica as mesmas operações de convolução, associando os mapas a conjuntos de *default boxes* com diferentes proporções de aspecto ($\{1.0, 2.0, 0.5, 3.0\}$) e escalas predefinidas ($[0.05, 0.15, 0.3, 0.45, 0.6, 0.75, 0.85, 1.0]$). Essa consistência estrutural garante que os *feature maps* de diferentes escalas sejam tratados de maneira uniforme, preservando a coerência na geração de predições.

Cada default box gerado pelo detector é associado a predições de classes e localizações de objetos. O número total de caixas geradas por escala é proporcional ao tamanho do feature map e ao número de proporções de aspecto especificadas. Essa configuração permite que o modelo mantenha sua capacidade de detecção em multiescala, mesmo nos ramos antecipados, alinhando-se com a arquitetura do modelo base original.

5.3 FLUXO DE TREINAMENTO COM ABORDAGEM SEQUENCIAL

O treinamento é dividido em três etapas principais. Na primeira, treina-se o modelo base utilizando apenas a saída final conectada ao detector principal. Durante essa fase, os pesos do *backbone* e do detector principal são otimizados para aprender representações globais robustas da base de dados, enquanto os pesos associados aos ramos antecipados (como *extra layers* e detectores dedicados) permanecem inativos. Essa estratégia garante que o modelo base estabeleça uma fundação sólida antes de introduzir as saídas antecipadas.

Concluída essa etapa, os pesos das camadas iniciais do *backbone* são congelados, preservando as representações aprendidas durante o treinamento inicial. Esse congelamento impede a necessidade de recalibrar gradientes em partes da rede já otimizadas, reduzindo o custo computacional e melhorando a estabilidade do processo de aprendizado. Além disso, assegura que as camadas compartilhadas forneçam informações consistentes para todos os ramos, mantendo a coerência entre os diferentes componentes do modelo.

Na última etapa, cada ramo antecipado é treinado de forma independente. Durante o treinamento de um ramo, apenas as camadas específicas desse ramo, incluindo as *extra layers* e o detector dedicado, são otimizadas, enquanto os pesos compartilhados do *backbone* permanecem inalterados. O processo envolve a inicialização do ramo e o ajuste de seus pesos com base

nas representações intermediárias geradas até o ponto de inserção. Dessa forma, cada ramo se especializa em escalas específicas de *feature maps*, evitando interferências e garantindo maior estabilidade ao aprendizado.

Por fim, esse fluxo de treinamento sequencial apresenta vantagens significativas, como eficiência computacional, especialização de componentes e independência entre os ramos. A preservação das representações globais permite redirecionar recursos para a especialização dos ramos antecipados, ao mesmo tempo em que a independência entre eles evita interferências mútuas. Assim, a abordagem assegura a consistência do modelo como um todo, pois as representações compartilhadas servem como alicerce para todas as saídas, mantendo a coerência entre os diferentes componentes.

Pseudocódigo do Fluxo de Treinamento

Algorithm 6 Treinamento Sequencial com Ramos Antecipados

Require: Conjunto de dados $\mathcal{D}$, épocas de treinamento para o modelo base E_{base},
Require: épocas de treinamento para cada ramo E_{branch}, taxa de aprendizado η

Etapa 1: Treinamento do modelo base
Inicializar o modelo base, $\theta_{\text{base}} \leftarrow$ `initialize_model_base()`
for $e \leftarrow 1 \dots E_{\text{base}}$ **do**
  for cada lote $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ **do**
    $\mathbf{f} \leftarrow$ `forward_backbone`$(\mathbf{x})$
    $\hat{\mathbf{y}} \leftarrow$ `forward_detector`$(\mathbf{f})$
    $\mathcal{L} \leftarrow$ `compute_loss`$(\hat{\mathbf{y}}, \mathbf{y})$
    $\theta_{\text{base}} \leftarrow \theta_{\text{base}} - \eta \nabla_{\theta_{\text{base}}} \mathcal{L}$
  end for
end for
Etapa 2: Congelamento de pesos compartilhados
`freeze_shared_backbone_weights()`
Etapa 3: Treinamento sequencial dos ramos antecipados
for cada ramo $r \in$ branches **do**
  $\theta_r \leftarrow$ `initialize_branch`(r)
  for $e \leftarrow 1 \dots E_{\text{branch}}$ **do**
    for cada lote $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ **do**
      $\mathbf{f}_{\text{shared}} \leftarrow$ `forward_backbone_shared`$(\mathbf{x})$
      $\mathbf{f}_r \leftarrow$ `forward_extra_layers`$(\mathbf{f}_{\text{shared}}, r)$
      $\hat{\mathbf{y}}_r \leftarrow$ `forward_branch_detector`$(\mathbf{f}_r, r)$
      $\mathcal{L}_r \leftarrow$ `compute_loss`$(\hat{\mathbf{y}}_r, \mathbf{y})$
      $\theta_r \leftarrow \theta_r - \eta \nabla_{\theta_r} \mathcal{L}_r$
    end for
  end for
end for

5.4 MÉTRICAS DE AVALIAÇÃO

Nesta seção, são descritas as métricas utilizadas para avaliar o desempenho dos modelos desenvolvidos.

As métricas de desempenho computacionais avaliadas para os modelos incluem:

- **Tempo de Inferência:** O tempo médio necessário para realizar o *forward pass* de uma entrada no modelo, excluindo etapas de pré-processamento. Antes da medição, o modelo realiza um aquecimento (*warm-up*) para estabilizar a GPU. Para cada entrada, sincroniza-se a GPU utilizando `torch.cuda.synchronize()` e registra-se o tempo total para múltiplas repetições. O tempo médio é calculado como a razão entre o tempo total e o número de repetições.

- **Latência:** Avaliada de maneira similar ao tempo de inferência, mas sem a etapa de *warm-up*. Cada execução é medida individualmente, registrando-se o tempo necessário para processar uma entrada. O valor médio é calculado a partir de múltiplas repetições.

- **Uso de GPU:** Medido como o pico de memória alocada durante o *forward pass*. Para isso, utiliza-se a função `torch.cuda.max_memory_allocated()`, que registra a maior quantidade de memória ocupada pela GPU durante a execução.

- **Operações de Ponto Flutuante por Segundo (FLOPs):** Calculadas utilizando a biblioteca `ptflops`. Os FLOPs representam o número total de operações de ponto flutuante realizadas pelo modelo para processar uma entrada. O número de parâmetros do modelo, que indica a quantidade de pesos treináveis na rede, também é obtido por meio desta biblioteca.

- **Potência Consumida:** Medida ao longo de um intervalo de tempo utilizando a biblioteca `pynvml`. A cada intervalo de amostragem, registra-se o consumo de energia da GPU em watts, calculando-se a média e o pico do consumo durante o período.

Essas métricas fornecem uma análise detalhada e quantitativa da eficiência computacional e do desempenho operacional dos modelos, permitindo uma comparação robusta entre eles e avaliando sua adequação para diferentes cenários práticos.

Para a tarefa de detecção de objetos, as métricas específicas utilizadas são a **Precisão Média** (*Mean Average Precision*, mAP) e o **Intersection over Union** (IoU). Ambas são

amplamente utilizadas na avaliação de modelos de detecção de objetos e fornecem uma visão detalhada sobre o desempenho do modelo em identificar corretamente objetos e localizar suas posições nas imagens.

A métrica **mAP** é uma medida de desempenho global, que avalia a precisão média ponderada do modelo em múltiplos limiares de IoU. Essa abordagem permite capturar tanto a capacidade do modelo de detectar objetos corretamente quanto sua habilidade de lidar com situações mais complexas, como objetos parcialmente ocluídos ou em escalas variadas. Ao utilizar múltiplos limiares, o *mAP* avalia o equilíbrio entre a precisão e a sensibilidade do modelo, garantindo que ele não dependa exclusivamente de um único critério de avaliação.

O **IoU**, por sua vez, é uma métrica que mede a sobreposição entre a região delimitada pelo modelo (bounding box predita) e a região real (ground truth). Embora sua definição matemática já tenha sido apresentada, sua aplicação prática reflete diretamente a qualidade da localização espacial dos objetos. Um alto valor de IoU indica que o modelo é capaz de gerar previsões que se alinham bem com os objetos reais, enquanto valores baixos sugerem erros na localização ou no dimensionamento das caixas preditas.

Em conjunto, essas métricas fornecem uma avaliação robusta do modelo, considerando tanto a precisão na classificação dos objetos quanto a qualidade da localização espacial. Enquanto o IoU avalia cada predição individualmente, o mAP sintetiza o desempenho global do modelo ao longo de diversas classes e limiares, oferecendo uma visão geral de sua eficácia.

No contexto do conjunto de dados KITTI, que apresenta desafios específicos como cenários complexos, oclusões e objetos em escalas variadas, o uso combinado de mAP e IoU é particularmente relevante. Essas métricas permitem não apenas avaliar o desempenho absoluto do modelo, mas também identificar possíveis limitações em cenários desafiadores, orientando ajustes futuros na arquitetura ou nos processos de treinamento.

Por fim, a análise conjunta dessas métricas com as demais, como o tempo de inferência e o uso de recursos computacionais, possibilita uma avaliação holística do modelo. Isso garante que o desempenho seja mensurado não apenas pela precisão na detecção, mas também pela eficiência computacional e adequação a aplicações práticas.

6 RESULTADOS E DISCUSSÃO

6.1 INTRODUÇÃO

Nesta seção, são apresentados os experimentos realizados para avaliar o desempenho do sistema proposto e de seus componentes em diferentes cenários, em comparação com o SSD com ResNet101 como modelo "base". Os experimentos foram projetados para investigar tanto a eficácia de cada módulo quanto o comportamento do sistema integrado. A base de dados foi utilizada em diferentes configurações, explorando a distinção de dificuldade entre os exemplos e o impacto dessa distinção no desempenho do modelo.

Os experimentos abordam:

- (5.3.1) o treinamento e avaliação do estimador de dificuldade.
- (5.3.2) A avaliação dos modelos base, e do modelo multi saídas, em termos de eficiência computacional.
- (5.3.3) A avaliação dos modelos base, e e do modelo multi saídas, treinados e testados na base de dados completa, sem distinção de dificuldade.
- (5.3.4) A avaliação dos modelos base, e do modelo multi saídas, treinados sem distinção de dificuldade e testados em cada classe de dificuldade.
- (5.3.5) A avaliação dos modelos base, e do modelo multi saídas, treinados com distinção de dificuldade e testados na classe de dificuldade em que foi treinado.
- (5.3.4) A avaliação do sistema completo.

(3) a avaliação das saídas do modelo dinâmico sem distinção de dificuldade, (4) a análise da relação entre os *scores* de dificuldade e o desempenho em diferentes classes de complexidade, (5) a especialização dos ramos do modelo dinâmico em conjuntos de dados específicos de dificuldade e (6) a avaliação do sistema completo integrando todas as partes do modelo.

6.2 CONFIGURAÇÃO DOS EXPERIMENTOS

6.2.1 Ambiente Experimental

O ambiente experimental foi configurado com uma máquina equipada com o sistema operacional Ubuntu 20.04, 64 GB de RAM e uma GPU NVIDIA RTX 3090, assegurando desempenho suficiente para o treinamento e avaliação de redes neurais complexas. O treinamento foi conduzido utilizando o framework PyTorch 2.0, com implementações customizadas para os modelos base e dinâmico.

6.2.2 Configuração dos Experimentos

Os experimentos foram realizados utilizando a base de dados *KITTI*, obtida e pré-processada por meio da plataforma Roboflow (KRAUSS, 2022). A base já foi disponibilizada com uma divisão fixa entre os conjuntos de treinamento, validação e teste, garantindo consistência entre os modelos avaliados. Os exemplos em cada conjunto são idênticos para todos os modelos testados, variando apenas os rótulos utilizados: o estimador de dificuldade utiliza os *scores* de dificuldade como rótulos, enquanto os modelos base e dinâmico utilizam as *bounding boxes* e as classes de objetos.

Os *scores* de dificuldade, utilizados como rótulos no treinamento do estimador, foram calculados automaticamente a partir de uma metodologia descrita previamente.

O pré-processamento incluiu o redimensionamento das imagens para uma resolução de 300×300, compatível com o SSD, além de transformações de dados como *random flipping* horizontal com probabilidade 0.5, ajustes aleatórios de brilho e contraste com probabilidade 0.2 e normalização utilizando os valores padrão do ImageNet (*mean* $= (0.485, 0.456, 0.406)$, *std* $= (0.229, 0.224, 0.225)$).

Além disso, cada subseção correspondente aos experimentos realizados detalha as configurações específicas, como taxa de aprendizado, otimizador e número de épocas, proporcionando clareza e reprodutibilidade. Essa organização modular permite compreender como as diferentes configurações afetam os resultados de cada experimento.

6.3 RESULTADOS DOS EXPERIMENTOS

6.3.1 Modelo Estimador de Dificuldades

O treinamento foi conduzido utilizando a base KITTI, com os escores de dificuldade produzidos pela metodologia anteriormente descrita, com transformações para aumentar a robustez do modelo, incluindo random resized cropping, horizontal flipping e normalização.

Os dados foram organizados em batches de tamanho 100, e o treinamento foi realizado utilizando o otimizador SGD com taxa de aprendizado inicial de 0.001, momento de 0.9 e fator de weight decay de 10^{-4}. O ajuste da taxa de aprendizado foi gerenciado por um agendador ReduceLROnPlateau, que reduzia a taxa em um fator de 0.1 sempre que a perda de validação não melhorava por 5 épocas consecutivas. O treinamento ocorreu por até 200 épocas, com um critério de early stopping de 30 épocas sem melhoria na perda de validação.

Para facilitar a rastreabilidade e análise do treinamento, foram utilizados MLflow e TensorBoard para registro de métricas e armazenamento dos modelos intermediários. O melhor modelo, determinado com base na menor perda de validação registrada, foi salvo automaticamente. Além disso, o estado do treinamento foi mantido para permitir retomadas caso necessário.

O modelo desenvolvido foi treinado para realizar uma tarefa de regressão, mapeando valores no intervalo contínuo $[0, 1]$. Para fins de análise de desempenho, os valores preditos foram discretizados em duas categorias: amostras com valores inferiores ou iguais a $0{,}5$ foram classificadas como pertencentes à classe "fácil", enquanto valores superiores a esse limiar foram atribuídos à classe "difícil". Essa estratégia permitiu uma avaliação do modelo sob uma perspectiva binária, facilitando a interpretação dos resultados em um contexto prático. Com essa separação, 448 de 747 imagens foram classificadas como fácil.

A abordagem adotada resultou em uma acurácia de 89%, evidenciando uma capacidade satisfatória do modelo em distinguir entre as duas classes definidas. Além disso, a eficiência computacional foi avaliada por meio da latência inferencial, a qual foi medida em $0{,}0009$ segundos por amostra. Cabe destacar que essa latência representa o tempo de processamento fundamental do modelo, sendo que quaisquer latências subsequentes em sistemas que utilizam essa predição já incluem esse tempo como base.

6.3.2 Estratégia Comparativa e Discussão dos Resultados

A avaliação comparativa foi estruturada para analisar a relação entre o desempenho em termos de *mAP* e as métricas de eficiência computacional (*latência*, *tempo de inferência*, *uso de memória GPU*, *parâmetros* e *FLOPs*) entre o modelo base (SSD original) e o modelo dinâmico com saídas antecipadas (*early exits*). A ideia central é compreender os trade-offs entre a redução do custo computacional e a perda de desempenho em detecção de objetos.

A comparação incluiu os seguintes modelos:

- **Modelo base (SSD original)**: Avaliado em todas as amostras, utilizando a saída final para predições.
- **Modelo dinâmico - Saída antecipada (ramo 1)**: Avaliado em todas as amostras, com foco na primeira saída intermediária.
- **Modelo dinâmico - Saída final (ramo 2)**: Avaliado em todas as amostras, representando o comportamento equivalente ao modelo base.

Cada modelo foi treinado a partir de pesos pré-treinados no ImageNet, uma taxa de aprendizado de 10^{-4}, *batch size* de 96, e por 400 épocas. Além disso, foi utilizada precisão mista (FP16) e o StepLR como *learning rate scheduler*. A *loss function* utilizada foi a padrão implementada no SSD do `torchvision`, incluindo a combinação de *Cross-Entropy Loss* para classificação e *Smooth L1 Loss* para regressão das *bounding boxes*. As métricas avaliadas incluem *mAP*, latência, tempo de inferência, uso de memória GPU, número de parâmetros e FLOPs.

O resultado esperado era que a saída antecipada (*ramo 1*) apresentasse uma redução significativa nas métricas de eficiência computacional (latência, memória e FLOPs), mas com uma perda considerável em *mAP*, devido à menor profundidade da rede até este ponto.

Tabela 11 – Média das métricas para o modelo Original e a saída antecipada, incluindo Max Memory GPU, e a diferença percentual.

Métrica	**Original - Saída final**	**Saída antecipada na 17a camada.**	**Dif. (%)**
Latência (s)	0.01493	0.01103	26.08%
InferenceTime (s)	0.01517	0.01096	27.77%
Max Mem GPU (MB)	540.26	314.17	41.89%
Consumo Energético (W)	174.57	154.80	11.33%
Params	52.97 M	21.68 M	59.07%
FLOPs (GMac)	18.00	11.61	35.50%

A Tabela 11 ilustra como a saída antecipada, inserida na 17ª camada do *SSD*, se compara ao modelo original (saída final) em termos de custo computacional. Observa-se uma redução significativa na latência (26,08%), no tempo de inferência (27,77%) e no uso de memória *GPU* (41,89%), além de uma diminuição de 11,33% no consumo energético. A queda no número de parâmetros (59,07%) e de *FLOPs* (35,50%) reforça a economia de recursos propiciada pela abordagem de *early exit*.

Esses resultados indicam que interromper o processamento mais cedo pode ser vantajoso em cenários com restrições de tempo de resposta, de memória ou até mesmo de consumo de energia, ainda que, potencialmente, haja impacto no *mAP* — aspecto que será comentado na análise subsequente sobre o desempenho em detecção de objetos. Dessa forma, fica evidente o *trade-off* entre preservar a precisão máxima do modelo (saída final) e promover uma execução mais leve (saída antecipada), cabendo ao projetista de sistemas decidir qual estratégia é mais adequada às exigências e limitações de cada aplicação.

6.3.3 Análise dos Resultados - Base de testes completa

Tabela 12 – Média das métricas para o modelo Original e o Truncado, incluindo Max Memory GPU, e a diferença percentual.

Métrica	**Base - Saída 2**	**Saída antecipada - Ramo 1**	**Dif. (%)**
Latência (s)	0.01493	0.01103	26.08%
Inference Time (s)	0.01517	0.01096	27.77%
Max Mem GPU (MB)	540.26	314.17	41.89%
Params	52.97 M	21.68 M	59.07%
FLOPs (GMac)	18.00	11.61	35.50%

Entretanto, essa redução veio acompanhada de uma perda considerável no desempenho de detecção, conforme mostrado na Tabela 14. O *mAP* para a saída antecipada (*ramo 1*) foi de apenas 29.45%, comparado a 56.07% do modelo base e da saída final (*ramo 2*). Esse resultado reflete a limitação das representações intermediárias extraídas em camadas mais rasas do *backbone*, que possuem menor capacidade de discriminação em comparação às representações geradas em camadas mais profundas.

Por outro lado, a saída final do modelo dinâmico (*ramo 2*) apresentou desempenho equi-

Tabela 14 – mAP e IoU para o modelo base e as saídas do modelo dinâmico.

Modelo	**Treinado em**	**Avaliado em**	***mAP***
Modelo Base	Base completa	Base completa	56.07%
Saída Final - Ramo 2	Base completa	Base completa	56.07%
Saída Antecipada - Ramo 1	Base completa	Base completa	29.45%

valente ao do modelo base, com *mAP* de 56.07%. Isso demonstra que as modificações introduzidas para suportar saídas antecipadas não impactaram negativamente a capacidade da saída final em comparação com o SSD original.

Esses resultados corroboram a expectativa inicial de que saídas antecipadas podem proporcionar ganhos substanciais de eficiência computacional, especialmente para aplicações onde trade-offs de desempenho em detecção são aceitáveis. A análise também reforça a importância de balancear a profundidade e a granularidade das representações ao projetar saídas antecipadas, de forma a minimizar perdas em *mAP*.

6.3.4 Análise dos Resultados por Nível de Dificuldade

O threshold de 0.4 para separar imagens fáceis de difíceis foi definido empiricamente, considerando a distribuição geral dos *scores*, conforme apresentado na Figura 42. Observa-se que a distribuição dos *scores* é aproximadamente simétrica, com a maior parte das amostras concentrada na faixa entre 0.3 e 0.5, indicando que a maioria das imagens apresenta uma dificuldade intermediária. O valor de 0.4 foi escolhido para segmentar as imagens de modo a separar um subconjunto significativo de exemplos mais simples (abaixo de 0.4) e exemplos mais complexos (acima de 0.4).

Essa divisão, embora empírica, permite avaliar o impacto do nível de dificuldade no desempenho dos modelos. Em trabalhos futuros, uma análise mais robusta poderá explorar técnicas de clusterização ou validação cruzada baseada no desempenho do modelo para determinar um *threshold* mais otimizado. No entanto, os resultados obtidos sugerem que a escolha de 0.4 é suficiente para validar a metodologia proposta e avaliar as diferenças de desempenho entre níveis de dificuldade.

A decisão de treinar os modelos na base completa, sem distinção prévia de dificuldade, tem como objetivo investigar como os modelos reagem a diferentes níveis de complexidade sem priorizar ou adaptar-se a um subconjunto específico durante o treinamento. Isso permite

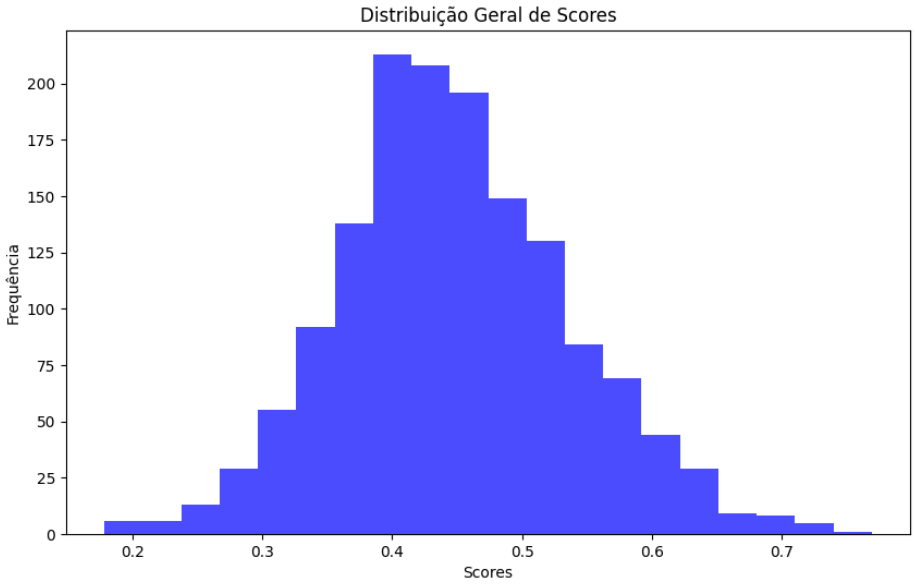


Figura 42 – Distribuição Geral de *Scores* de Dificuldade.

observar as limitações e vantagens de cada saída do modelo dinâmico em relação ao modelo base, avaliando sua capacidade de generalização.

Treinar na base completa também fornece um ponto de partida para comparações futuras, onde os ramos do modelo dinâmico serão especializados em subconjuntos específicos (imagens fáceis ou difíceis). Essa abordagem sequencial demonstra como a introdução da distinção de dificuldade pode impactar o desempenho geral e a eficiência do sistema.

A Tabela 15 apresenta os valores de *mAP* para o modelo base e para os dois ramos do modelo dinâmico, avaliados separadamente nos subconjuntos de imagens fáceis (448 amostras) e difíceis (299 amostras). É importante destacar que os modelos foram treinados na base completa, sem distinção de dificuldade, e posteriormente avaliados nos subconjuntos.

Tabela 15 – *mAP* para o modelo base e as saídas do modelo dinâmico, avaliados separadamente nos subconjuntos de imagens fáceis e difíceis.

Modelo	**Imagens Fáceis (%)**	**Imagens Difíceis (%)**
Modelo **SSD300 Original**	65.85	43.64
Modelo Proposto – (Ramo Final Especializado)	–	55.40
Modelo Proposto – (Ramo Antecipado Especializado)	54.81	–

Os resultados mostram que o ramo antecipado (*ramo 1*) apresenta um desempenho inferior tanto para imagens fáceis quanto para imagens difíceis, enquanto o ramo final (*ramo 2*) mantém um desempenho semelhante ao modelo base. Essa discrepância é justificada pela arquitetura do modelo: o ramo antecipado é menos profundo, extraindo representações mais

6.3.5 Treinamento e Avaliação dos Ramos Especializados

Neste experimento, cada ramo do modelo dinâmico foi treinado separadamente nos subconjuntos de imagens fáceis e difíceis, enquanto o *backbone* comum permaneceu congelado para preservar as representações gerais aprendidas no treinamento inicial. A divisão das imagens nos dois subconjuntos foi realizada utilizando o threshold de 0.4, definido empiricamente conforme discutido anteriormente. A especialização permitiu ajustar os pesos dos ramos para as características específicas de cada nível de dificuldade, visando validar a eficácia da abordagem proposta.

Durante o *fine-tuning*, foi utilizada uma taxa de aprendizado reduzida em 1e-2, uma estratégia essencial para evitar grandes perturbações nos pesos, garantindo que o modelo convergisse adequadamente para as características específicas de cada subconjunto sem superespecializar-se.

Tabela 17 – *mAP* para os ramos especializados após *fine-tuning* nos subconjuntos de imagens fáceis e difíceis.

Modelo	**Imagens Fáceis (%)**	**Imagens Difíceis (%)**
Modelo SSD300 Original	65.85	43.64
Modelo Proposto – Saída Final (Ramo 2 Especializado)	–	55.40
Modelo Proposto – Saída Antecipada (Ramo 1 Especializado)	54.81	–

Os resultados apresentados na Tabela 17 mostram que a especialização dos ramos levou a melhorias significativas no *mAP*, particularmente no ramo dedicado às imagens difíceis. Para este ramo, o desempenho aumentou de 43.64% para 55.40%, quando comparado ao modelo treinado na base completa e avaliado apenas nas imagens difíceis. Esse resultado era esperado, pois o treinamento agora se concentrou exclusivamente nas imagens mais complexas, permitindo um ajuste mais refinado dos pesos. Por outro lado, o ramo especializado em imagens fáceis apresentou uma queda no *mAP*, de 65.85% para 54.81%. Essa redução, embora notável, é menor do que a observada para o ramo antecipado no experimento anterior, indicando que as representações mais simples exigidas pelas imagens fáceis são adequadamente ajustadas durante o *fine-tuning*.

Esses resultados reforçam a validade da abordagem proposta de divisão do treinamento por dificuldade. O aumento do *mAP* para os ramos especializados valida a hipótese de que a separação dos dados por níveis de dificuldade permite modelos mais eficazes e adaptados às características específicas de cada classe de dificuldade.

Esses resultados indicam que a especialização no treinamento é uma estratégia eficaz para aumentar o desempenho em cenários específicos, como mostrado pelas melhorias substanciais nos ramos dedicados às imagens difíceis. Contudo, a análise limitada às imagens fáceis e difíceis destaca a necessidade de estratégias adicionais para lidar com exemplos de dificuldade intermediária, tornando este um aspecto importante para futuros trabalhos.

6.3.6 Resultados do Sistema Completo

O sistema completo, que incorpora a política de decisão com base nos *scores* de dificuldade, apresentou resultados que equilibram eficiência computacional e desempenho em detecção de objetos. A Tabela 19 resume as métricas avaliadas, incluindo *mAP*, *IoU*, consumo médio de memória GPU, tempo médio de inferência e consumo médio de energia.

Tabela 19 – mAP, IoU e métricas de eficiência computacional para o sistema completo.

Modelo	***mAP***	**Memória GPU (MB)**	**Tempo de Inferência (s)**	**Potência (W)**
Modelo Base (SSD300 original)	56,07%	540,26	0,01517	174,57
Sistema Dinâmico Proposto	55,16%	448,76	0,01349	149,94
Ganho (%)	**-1%**	**16,75%**	**11,10%**	**14,09%**

Os resultados indicam uma leve redução de 1% no *mAP* em relação ao modelo base, enquanto as métricas de eficiência computacional apresentaram melhorias significativas: redução de $16,75\%$ no uso de memória GPU, $11,10\%$ no tempo de inferência e $14,09\%$ no consumo de energia. Esses ganhos justificam a adoção do sistema dinâmico em aplicações onde a eficiência computacional é uma prioridade.

A redução no *mAP* era esperada, dado que os ramos individuais apresentaram desempenho ligeiramente inferior ao modelo base. Esse resultado pode ser atribuído à menor profundidade do ramo antecipado (*ramo 1*) e à política de decisão, que, embora eficiente, pode ocasionalmente alocar imagens de maior dificuldade para ramos menos adequados. Essa limitação reforça a necessidade de melhorias na política de decisão, como a análise mais aprofundada do impacto do threshold utilizado.

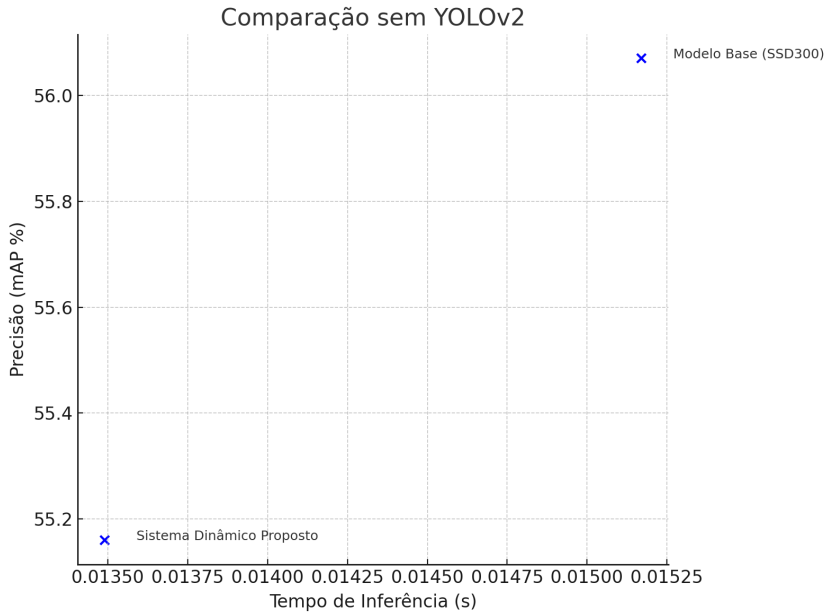


Figura 43 – Comparação entre modelo base e o sistema proposto. Figura elaborada pelo autor.

No entanto, os ganhos nas métricas de eficiência são substanciais e demonstram que a abordagem proposta é válida para cenários com restrições de recursos computacionais, como sistemas embarcados e aplicações em tempo real. Além disso, a leve perda de *mAP* sugere que há espaço para otimizações adicionais, tanto na arquitetura quanto na política de decisão, para potencializar os benefícios sem comprometer o desempenho.

A proporção de imagens classificadas como fáceis e difíceis foi analisada anteriormente, com o threshold de $0,4$ sendo consistente em todos os experimentos. Ajustes nesse threshold poderiam influenciar as métricas finais, mas essa análise foi deixada como trabalho futuro, pois foge do escopo atual.

Por fim, os resultados apresentados sugerem que o sistema completo é robusto e eficiente, com potencial para ser implementado em aplicações práticas. Contudo, sua generalização para bases de dados mais diversificadas e para imagens de dificuldade intermediária ainda requer investigações adicionais, que serão abordadas em trabalhos futuros.

7 CONCLUSÃO E TRABALHOS FUTUROS

7.1 INTRODUÇÃO

Neste capítulo, são apresentados os principais resultados alcançados ao longo deste trabalho, juntamente com reflexões sobre as contribuições realizadas e suas implicações para o campo de estudo. Além disso, são discutidas as limitações identificadas no decorrer do desenvolvimento, bem como possíveis direções para trabalhos futuros que possam expandir ou aprimorar os métodos e abordagens aqui propostos. A conclusão visa sintetizar os achados mais relevantes, contextualizá-los dentro do objetivo geral do projeto e destacar o impacto potencial das soluções desenvolvidas em cenários reais de aplicação.

7.2 CONCLUSÕES

Este trabalho explorou o uso de redes neurais dinâmicas aplicadas à detecção de objetos, introduzindo saídas antecipadas para equilibrar precisão e eficiência computacional. O modelo proposto foi baseado no *Single Shot Detector* (SSD), adaptado para incluir ramificações (*branches*) capazes de processar imagens com diferentes níveis de dificuldade. A integração de um estimador de dificuldade e uma política de decisão adaptativa formou o núcleo da solução apresentada.

Os experimentos conduzidos validaram a eficácia da abordagem sob múltiplos aspectos. Os principais resultados mostram que o sistema dinâmico alcançou uma redução significativa em métricas de eficiência computacional, como uso de memória da GPU, consumo energético e tempo de inferência, mantendo uma redução de apenas 1% no *mAP* em relação ao modelo base. Esse equilíbrio evidencia o potencial do sistema para aplicações em cenários onde a eficiência computacional é crítica, como em dispositivos embarcados e sistemas autônomos.

Além disso, os experimentos com os ramos especializados reforçaram a eficácia da abordagem de segmentação por dificuldade. Quando os ramos foram treinados e avaliados em subconjuntos específicos de imagens fáceis e difíceis, observou-se um aumento considerável no *mAP*, demonstrando que a especialização contribui para o desempenho do sistema.

Por outro lado, o trabalho também revelou limitações importantes. A política de decisão, embora funcional, ainda pode ser aprimorada, especialmente para lidar com imagens de dificuldade intermediária. Além disso, ajustes no *threshold* de classificação podem trazer ganhos

adicionais, mas essa análise não foi explorada devido à sua complexidade, sendo sugerida como trabalho futuro.

Em termos metodológicos, a pesquisa destacou a viabilidade de medir e utilizar a dificuldade das imagens para guiar decisões de inferência, apontando para uma nova direção no desenvolvimento de redes neurais dinâmicas. A integração da métrica de dificuldade e a adaptação da arquitetura SSD para suportar saídas antecipadas foram conquistas fundamentais que abrem caminho para novas investigações nesse campo.

Por fim, os resultados obtidos não apenas validam a abordagem proposta, mas também sugerem que os ganhos de eficiência computacional podem ser ampliados com ajustes adicionais na arquitetura, nas políticas de decisão e no treinamento. Este trabalho, portanto, contribui significativamente para a aplicação de redes neurais dinâmicas em detecção de objetos, oferecendo um ponto de partida robusto para futuras pesquisas.

7.2.1 Resumo das Contribuições

Este trabalho apresenta contribuições significativas no campo de redes neurais dinâmicas aplicadas à detecção de objetos, abrangendo aspectos metodológicos, arquiteturais e experimentais. A seguir, resumem-se as principais contribuições:

- **Proposição de Early-Exit para Detecção de Objetos:** Este trabalho apresenta a segunda proposta conhecida de implementação de redes neurais dinâmicas com saídas antecipadas (*early-exit*) aplicada à detecção de objetos. A extensão para essa tarefa é desafiadora devido à sua complexidade inerente, e o estudo contribui significativamente ao adaptar o paradigma para o modelo SSD.

- **Desenvolvimento de uma Métrica de Dificuldade:** Foi introduzida uma métrica inédita para quantificar a dificuldade de imagens em tarefas de detecção de objetos. Essa métrica, baseada no *ground truth*, guia decisões estratégicas de inferência, evidenciando a importância de considerar a complexidade das entradas.

- **Integração de Estimativa de Dificuldade e Decisão Adaptativa:** Além da métrica de dificuldade, foi desenvolvido um modelo auxiliar que estima esse escore durante a inferência, com baixo custo computacional. Este estimador é integrado a uma política adaptativa que decide dinamicamente em qual saída processar cada imagem.

- **Adaptação do SSD com Ramificações de Early-Exit:** O trabalho modifica a arquitetura original do SSD para incluir saídas antecipadas, detalhando as mudanças estruturais e metodológicas necessárias para sua implementação.
- **Análise de Estratégias de Treinamento:** Foram revisadas e implementadas diferentes estratégias de treinamento para redes multi-exit, abordando os desafios específicos de integrar ramificações de saída e políticas adaptativas em modelos de detecção de objetos.
- **Validação Experimental Completa:** Uma ampla gama de experimentos valida a eficácia da abordagem proposta, considerando tanto métricas de precisão (*mAP*, *IoU*) quanto de eficiência computacional (tempo de inferência, consumo de memória e energia). Os resultados demonstram a viabilidade do sistema para aplicações práticas, como veículos autônomos e dispositivos embarcados.
- **Identificação de Limitações e Direções Futuras:** São discutidas as limitações da abordagem, como a necessidade de ajustes mais robustos na política de decisão e a investigação de imagens com dificuldade intermediária, além de sugerir possíveis melhorias e novas áreas de pesquisa.

7.3 LIMITAÇÕES DO ESTUDO

Apesar dos avanços apresentados, este trabalho possui algumas limitações importantes que devem ser consideradas para futuras pesquisas. Primeiramente, a política de decisão utilizada para determinar a saída antecipada foi implementada com base em um *threshold* fixo de dificuldade (0.4). Embora funcional, essa abordagem pode ser melhorada para considerar diferentes critérios adaptativos ou aprender diretamente a partir dos dados. O impacto de ajustar o *threshold* sobre o desempenho do sistema não foi analisado, e tal investigação poderia revelar oportunidades para otimização.

Outro ponto a ser destacado é a ausência de uma análise aprofundada sobre o desempenho do sistema ao lidar com imagens de dificuldade intermediária, que não se enquadram claramente como "fáceis"ou "difíceis". Esse cenário representa um desafio prático em aplicações reais, onde as distribuições de dificuldade podem variar significativamente em relação ao conjunto de treinamento.

Além disso, o modelo dinâmico demonstrou limitações em termos de *mAP* para ramos individuais, especialmente no caso do ramo antecipado, devido à menor profundidade e capacidade representacional. Embora esperado, esse comportamento evidencia a necessidade de explorar arquiteturas mais robustas para esses ramos.

Finalmente, a abordagem proposta foi avaliada em um conjunto de dados específico com base no formato COCO, que foi adaptado para o problema. O comportamento do sistema em bases mais diversificadas, com distribuições distintas de dificuldade e características visuais, não foi explorado. Essa limitação representa uma oportunidade de validação externa e generalização do método em outros contextos.

Essas limitações sugerem a necessidade de investigações futuras que aprimorem tanto a política de decisão quanto a arquitetura dos ramos, além de análises mais amplas sobre a aplicabilidade do sistema em cenários mais variados e realistas.

7.4 TRABALHOS FUTUROS

- **Estudo Crítico da Quantidade e Posição dos Ramos:** Realizar análises detalhadas para determinar o número ideal de ramos de saída antecipada (*early-exit*) e suas posições na arquitetura, investigando como essas escolhas impactam o equilíbrio entre eficiência computacional e desempenho em detecção de objetos.

- **Estudo Comparativo entre Métodos de Treinamento:** Comparar diferentes estratégias de treinamento, como o treinamento conjunto (*joint training*) e o treinamento sequencial (*sequential training*), avaliando suas vantagens e limitações em termos de desempenho e eficiência computacional.

- **Meta-Aprendizado para Níveis de Dificuldade:** Explorar técnicas de meta-aprendizado para ajustar automaticamente os níveis de dificuldade, permitindo que o sistema adapte os escores de dificuldade às características específicas dos dados de entrada.

- **Ajuste Automático de Thresholds:** Investigar estratégias para ajustar dinamicamente os *thresholds* de decisão da política, buscando otimizar tanto a precisão quanto a eficiência computacional. Essa análise pode incluir estudos sobre a sensibilidade do sistema a diferentes valores de *threshold*.

- **Avaliação em Cenários Reais:** Validar o sistema em cenários práticos, como veículos autônomos e dispositivos móveis, para avaliar sua eficácia em condições reais de uso, considerando restrições de tempo, energia e recursos de hardware.
- **Avaliação de Imagens de Dificuldade Intermediária:** Realizar estudos sobre o comportamento do sistema ao lidar com imagens de dificuldade intermediária, que não se enquadram claramente como "fáceis"ou "difíceis". Isso pode incluir ajustes na política de decisão para considerar níveis intermediários de dificuldade.
- **Exploração de Arquiteturas Alternativas:** Testar outras arquiteturas de redes neurais para os ramos antecipados, como MobileNet ou EfficientNet, buscando alcançar um melhor equilíbrio entre profundidade, capacidade representacional e custo computacional.
- **Otimização de Eficiência Computacional:** Investigar técnicas adicionais, como poda de modelos (*pruning*) ou quantização, para reduzir ainda mais o consumo de recursos computacionais sem comprometer significativamente o desempenho do sistema.
- **Impacto de Thresholds Adaptativos:** Explorar como diferentes *thresholds* impactam o desempenho geral do sistema, tanto em termos de eficiência computacional quanto de precisão, e investigar estratégias de ajuste automático desses valores durante a inferência.
- **Análise Comparativa de Impactos:** Realizar uma análise detalhada do impacto de diferentes decisões arquiteturais e metodológicas no contexto de detecção de objetos, considerando métricas como *mAP*, tempo de inferência, consumo energético e uso de memória da GPU.
- **Testes em Outras Bases de Dados:** Replicar o experimentos em bases de dados populares de detecção de objeto, como COCO e PascalVOC.
- **Comparação com Trabalhos Relacionados:** Foi solicitado acesso ao código dos projetos listados em trabalhos relacionados, porém não houve retorno por parte dos autores. O trabalho futuro consistiria em implementar os artigos para fins comparativos.

7.5 CONSIDERAÇÕES FINAIS

Este trabalho apresentou uma abordagem inovadora para a detecção de objetos utilizando redes neurais dinâmicas com saídas antecipadas. Apesar de ser um campo pouco explorado,

especialmente no contexto de detecção de objetos, a investigação aqui realizada se destaca por abordar tanto os desafios técnicos quanto a validação prática da metodologia. Na literatura existente, apenas um trabalho aplica saídas antecipadas em modelos de detecção, mas ele não detalha os desafios técnicos envolvidos nessa tarefa, tampouco oferece uma análise abrangente sobre os impactos dessa abordagem em termos de eficiência computacional e precisão.

Os resultados obtidos foram satisfatórios, demonstrando que a introdução de saídas antecipadas, combinada com um estimador de dificuldade e uma política de decisão adaptativa, pode reduzir significativamente o consumo de recursos computacionais sem comprometer drasticamente o desempenho do modelo em termos de *mAP*. Contudo, a análise também revelou que ainda há muito espaço para melhorias.

Limitações como a sensibilidade ao *threshold* de dificuldade, o desempenho em imagens de dificuldade intermediária e a possibilidade de otimizações adicionais nos ramos e na política de decisão indicam que o sistema proposto é apenas um ponto de partida. Avanços futuros podem incluir o desenvolvimento de políticas de decisão mais sofisticadas, ajustes automáticos de *thresholds*, a exploração de arquiteturas alternativas para os ramos e a validação do sistema em cenários mais variados e realistas.

Portanto, este trabalho não apenas contribui para o avanço do estado da arte em redes neurais dinâmicas aplicadas à detecção de objetos, mas também lança as bases para abordagens mais sofisticadas que possam explorar todo o potencial desse paradigma. Acredita-se que os resultados apresentados servirão como inspiração e referência para futuras pesquisas, promovendo o desenvolvimento de sistemas mais adaptáveis, eficientes e robustos no campo da visão computacional.

REFERÊNCIAS

ALCANTARA, L. A. de M. *Aprendizagem Contínua para Classificação de Imagens*. Dissertação (Mestrado) — Universidade Federal de Pernambuco, Recife, Brazil, 2024. Dissertação (Mestrado) – Programa de Pós-graduação em Ciência da Computação.

ALRASHIDE, I.; ALKHALIFAH, H.; AL-MOMEN, A.-A.; ALALI, I.; ALSHAIKH, G.; RAHMAN, A.; SAADELDEEN, A.; ALOUP, K. Aims: Ai based mental healthcare system. v. 23, p. 225–234, 12 2023.

BEDNY, M.; RICHARDSON, H.; SAXE, R. Visual cortex responds to spoken language in blind children. *The Journal of Neuroscience*, Society for Neuroscience, v. 35, n. 33, p. 11674–11681, 2015.

BENGIO, Y. Practical recommendations for gradient-based training of deep architectures. *Neural Networks: Tricks of the Trade*, v. 7700, p. 437–478, 2012.

BIEDERMAN, I. Recognition-by-components: A theory of human image understanding. *Psychological review*, American Psychological Association, v. 94, n. 2, p. 115, 1987.

BISHOP, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006. Disponível em: <https://www.springer.com/gp/book/9780387310732>.

BRILLIANT.ORG. *Feedforward Neural Networks*. 2024. Acessado em: jun. 2024. Disponível em: <https://brilliant.org/wiki/feedforward-neural-networks/>.

BRUCE, N. D.; TSOTSOS, J. K. Saliency, attention, and visual search: An information theoretic approach. *Journal of Vision*, The Association for Research in Vision and Ophthalmology, v. 9, n. 3, p. 5–5, 2009.

CHEN, L.-C.; PAPANDREOU, G.; KOKKINOS, I.; MURPHY, K.; YUILLE, A. L. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

CHEN, S.; CHEN-SONG, D. Detection, recognition, and tracking: A survey. *arXiv preprint arXiv:2203.11900*, 2022.

CHENG, Y.; WANG, D.; ZHOU, P.; ZHANG, T. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017. Disponível em: <https://arxiv.org>.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In: IEEE. *2009 IEEE conference on computer vision and pattern recognition*. [S.l.], 2009. p. 248–255.

DRäGER, S.; DUNKELAU, J. Evaluating the impact of loss function variation in deep learning for classification. *arXiv preprint arXiv:2210.16003*, 2022. Disponível em: <http://arxiv.org/abs/2210.16003v1>.

DUCHOWSKI, A. T.; GEISLER, W. S.; NOWROUZEZAHRAI, D.; SUNDSTEDT, V. Using microsaccades to estimate visual task difficulty in layered displays. *Journal of Vision*, The Association for Research in Vision and Ophthalmology, v. 19, n. 4, p. 3–3, 2019.

DUNCAN, J.; HUMPHREYS, G. W. Visual search and stimulus similarity. *Psychological Review*, American Psychological Association, v. 96, n. 3, p. 433–458, 1989.

FIGURNOV, M.; COLLINS, M.; ZHU, Y.; ZHANG, L.; HUANG, J.; VETROV, D.; SALAKHUTDINOV, R. Spatially adaptive computation time for residual networks. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2017. p. 1039–1048.

FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, Elsevier, v. 55, n. 1, p. 119–139, 1997.

GeeksforGeeks. *Multi-Layer Perceptron Learning in TensorFlow*. 2024. Disponível em: <https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/>.

GEIGER, A.; LENZ, P.; URTASUN, R. Are we ready for autonomous driving? the kitti vision benchmark suite. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2012. p. 3354–3361.

GIRSHICK, R. Fast r-cnn. In: IEEE. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. [S.l.], 2015. p. 1440–1448.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. MIT Press, 2016. Disponível em: <https://www.deeplearningbook.org/>.

GRANDHE, R. *A Brief Guide to Post-Processing Methods for Object Detection*. 2021. Acessado em: 3 fev. 2025. Disponível em: <https://rushalig24.medium.com/a-brief-guide-to-post-processing-methods-for-object-detection-6c695ca2f11>.

HAFIZ, A. M.; BHAT, G. M. A survey on instance segmentation: State of the art. *arXiv preprint arXiv:2007.00047*, 2020.

HAMMER, B.; VILLMANN, T. Mathematical aspects of neural networks. In: . [S.l.: s.n.], 2003. p. 59–72.

HAN, S.; MAO, H.; DALLY, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In: *International Conference on Learning Representations (ICLR)*. [s.n.], 2016. ArXiv:1510.00149. Disponível em: <https://arxiv.org/abs/1510.00149>.

HAN, Y.; HUANG, G.; SONG, S.; YANG, L.; WANG, H.; WANG, Y. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, v. 44, n. 11, p. 7436–7456, 2021.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd. ed. New York: Springer, 2009. ISBN 978-0-387-84857-0.

HE, K.; GKIOXARI, G.; DOLLÁR, P.; GIRSHICK, R. Mask r-cnn. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, p. 2961–2969, 2017.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2016. p. 770–778.

HECTOR, J.; YUAN, W.; LEE, J. Scalable object detection for edge computing using elastic neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, IEEE, v. 32, n. 7, p. 3012–3024, 2021.

HINTON, G. *Lecture 6e: RMSProp: Divide the gradient by a running average of its recent magnitude*. 2012. Coursera: Neural Networks for Machine Learning. Video lectures.

HINTON, G.; VINYALS, O.; DEAN, J. *Distilling the knowledge in a neural network*. 2015. ArXiv:1503.02531. Disponível em: <https://arxiv.org/abs/1503.02531>.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.

HOOGE, I. T.; ERKELENS, C. J. Adjustment of fixation duration in visual search. *Vision Research*, Elsevier, v. 38, n. 9, p. 1295–1302, 1998.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural networks*, Elsevier, v. 2, n. 5, p. 359–366, 1989.

HOWARD, A. G.; ZHU, M.; CHEN, B.; KALENICHENKO, D.; WANG, W.; WEYAND, T.; ANDREETTO, M.; ADAM, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*, 2017.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International conference on machine learning*, PMLR, p. 448–456, 2015.

IONESCU, R. T.; FOTEINI, L.; POPESCU, M.; IONESCU, B. A. How hard can it be? estimating the difficulty of visual search in an image. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2016. p. 2157–2166.

JIAO, L.; ZHANG, F.; LIU, F.; YANG, S.; LI, L.; FENG, Z.; QU, R. A survey of deep learning-based object detection. *IEEE Transactions on Neural Networks and Learning Systems*, IEEE, v. 31, n. 7, p. 2837–2852, 2020.

JOUPPI, N. P.; YOUNG, C.; PATIL, N.; PATTERSON, D.; AGRAWAL, G.; BAJWA, R.; BATES, S.; BHATIA, S.; BODEN, N.; BORCHERS, A. et al. In-datacenter performance analysis of a tensor processing unit. In: IEEE. *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. [S.l.], 2017. p. 1–12.

KANDEL, E. R.; SCHWARTZ, J. H.; JESSELL, T. M. *Principles of Neural Science*. [S.l.]: McGraw-Hill, 2000.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

KRAUSS, S. Open Source Dataset, *KITTI Dataset*. Roboflow, 2022. <https://universe.roboflow.com/sebastian-krauss/kitti-9amcz>. Visited on 2025-01-18. Disponível em: <https://universe.roboflow.com/sebastian-krauss/kitti-9amcz>.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, AcM New York, NY, USA, v. 60, n. 6, p. 84–90, 2012.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2012. v. 25, p. 1097–1105.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.

LECUN, Y.; BOSER, B. E.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKEL, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation*, MIT Press, v. 1, n. 4, p. 541–551, 1989.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, IEEE, v. 86, n. 11, p. 2278–2324, 1998.

LECUN, Y.; RANZATO, M. Efficient learning of sparse representations with an energy-based model. *Proceedings of the International Conference on Machine Learning (ICML)*, 2012.

LIN, T.-Y.; DOLLÁR, P.; GIRSHICK, R.; HE, K.; HARIHARAN, B.; BELONGIE, S. Feature pyramid networks for object detection. In: IEEE. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.], 2017. p. 2117–2125.

LIN, T.-Y.; DOLLáR, P.; GIRSHICK, R.; HE, K.; HARIHARAN, B.; BELONGIE, S. Feature pyramid networks for object detection. *arXiv preprint arXiv:1612.03144*, 2017. Acessado em: 3 fev. 2025. Disponível em: <https://arxiv.org/abs/1612.03144>.

LIN, T.-Y.; GOYAL, P.; GIRSHICK, R.; HE, K.; DOLLÁR, P. Focal loss for dense object detection. In: IEEE. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. [S.l.], 2017. p. 2980–2988.

LIN, T.-Y.; MAIRE, M.; BELONGIE, S.; HAYS, J.; PERONA, P.; RAMANAN, D.; DOLLÁR, P.; ZITNICK, C. L. Microsoft coco: Common objects in context. In: SPRINGER. *Proceedings of the European Conference on Computer Vision (ECCV)*. [S.l.], 2014. p. 740–755.

LIN, Z.; WANG, Y.; ZHANG, J.; CHU, X. Dynamicdet: A unified dynamic architecture for object detection. *arXiv preprint arXiv:2304.05552*, 2023.

LIU, W.; ANGUELOV, D.; ERHAN, D.; SZEGEDY, C.; REED, S.; FU, C.-Y.; BERG, A. C. Ssd: Single shot multibox detector. In: *European Conference on Computer Vision*. [S.l.]: Springer, 2016. p. 21–37.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, n. 4, p. 115–133, 1943.

MICIKEVICIUS, P.; NARANG, S.; ALBEN, J.; DIAMOS, G. F.; ELSEN, E.; GARCÍA, D.; GINSBURG, B.; HOUSTON, M.; KUCHAIEV, O.; VENKATESH, G.; WU, H. Mixed precision training. *CoRR*, abs/1710.03740, 2017. Disponível em: <http://arxiv.org/abs/1710.03740>.

MINSKY, M.; PAPERT, S. *Perceptrons: An introduction to computational geometry*. [S.l.]: MIT Press, 1969.

MNIH, V.; HEESS, N.; GRAVES, A. Recurrent models of visual attention. In: *Advances in Neural Information Processing Systems (NeurIPS)*. [S.l.: s.n.], 2014. p. 2204–2212.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. [s.n.], 2010. p. 807–814. Disponível em: <http://www.cs.toronto.edu/~hinton/absps/reluICML.pdf>.

NG, A. Y. Feature selection, l1 vs. l2 regularization, and rotational invariance. *Proceedings of the 21st International Conference on Machine Learning (ICML)*, p. 78, 2004.

NGUYEN, T.-H.; SCHERER, R.; LE, V.-H. Yolo series for human hand action detection and classification from egocentric videos. *Sensors*, v. 23, p. 3255, 03 2023.

P, H. R.; SRIVASTAVA, V.; CHAURASIA, K.; PACHECO, R. G.; COUTO, R. S. Early-exit deep neural network - a comprehensive survey. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 57, n. 3, nov. 2024. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3698767>.

PAPERT, S. *Summer vision project*. 1966.

PASCANU, R.; MIKOLOV, T.; BENGIO, Y. On the difficulty of training recurrent neural networks. *International Conference on Machine Learning (ICML)*, p. 1310–1318, 2013.

POMPLUN, M.; REINGOLD, E. M.; SHEN, J. Investigating attentional control in visual search: An individual differences approach. *Psychonomic Bulletin & Review*, Springer, v. 8, n. 4, p. 676–682, 2001.

PORTER, G.; TROSCIANKO, T. Effort during visual search and counting: Insights from pupilometry. *Vision Research*, Elsevier, v. 47, n. 16, p. 2134–2142, 2007.

PRECHELT, L. Early stopping - but when? *Neural Networks: Tricks of the Trade*, v. 1524, p. 55–69, 1998.

RADEčIć, D. *Softmax Activation Function Explained*. 2024. Disponível em: <https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60>.

REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2016. p. 779–788.

REN, S.; HE, K.; GIRSHICK, R.; SUN, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2015. v. 28, p. 91–99.

RENSINK, R. A.; ENNS, J. T. Preemption effects in visual search: Evidence from selective adaptation to motion and texture. *Perception & Psychophysics*, Springer, v. 57, n. 6, p. 853–864, 1995.

ROBBINS, H.; MONRO, S. A stochastic approximation method. *The Annals of Mathematical Statistics*, Institute of Mathematical Statistics, v. 22, n. 3, p. 400–407, 1951.

ROBERTS, L. G. *Machine Perception of Three-Dimensional Solids*. Tese (Doutorado) — Massachusetts Institute of Technology, 1963.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, American Psychological Association, v. 65, n. 6, p. 386–408, 1958.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group UK London, v. 323, n. 6088, p. 533–536, 1986. Disponível em: <https://www.nature.com/articles/323533a0>.

RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATHY, A.; KHOSLA, A.; BERNSTEIN, M. et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, Springer, v. 115, n. 3, p. 211–252, 2015.

SAHA, S. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. 2018. Acessado em: jun. 2024. Disponível em: <https://medium.com/towards-data-science/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>.

SALAU, A.; JAIN, S. Feature extraction: A survey of the types, techniques, applications. *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, IEEE, p. 330–335, 2020.

SENIOR, A.; HEIGOLD, G.; YANG, K.; BENGIO, S. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, v. 29, n. 6, p. 82–97, 2013.

SHEN, X. A survey of object classification and detection based on 2d/3d data. *arXiv preprint arXiv:1905.12683*, 2019.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

SMITH, L. N. No more pesky learning rate guessing games. *CoRR*, abs/1506.01186, 2015. Disponível em: <http://arxiv.org/abs/1506.01186>.

SMITH, L. N. Cyclical learning rates for training neural networks. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, p. 464–472, 2017.

SOVIANY, P.; IONESCU, R. T. Optimizing the trade-off between single-stage and two-stage object detectors using image difficulty prediction. In: *Proceedings of the IEEE International Conference on Computer Vision Workshops (ICCVW)*. [S.l.: s.n.], 2018. p. 2078–2086.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. *The journal of machine learning research*, JMLR.org, v. 15, n. 1, p. 1929–1958, 2014.

TAN, M.; LE, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. [S.l.: s.n.], 2019.

TEERAPITTAYANON, S.; MCDANEL, B.; KUNG, H. Branchynet: Fast inference via early exiting from deep neural networks. In: IEEE. *International Conference on Pattern Recognition (ICPR)*. [S.l.], 2016. p. 2464–2469.

TEERAPITTAYANON, S.; MCDANEL, B.; KUNG, H. T. *BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks*. 2017. Disponível em: <https://arxiv.org/abs/1709.01686>.

TIAN, F.; LUO, J.; YANG, M.; HUA, G. Query-dependent estimation of visual search difficulty using reconstruction error. *IEEE Transactions on Cybernetics*, IEEE, v. 44, n. 11, p. 2243–2252, 2014.

Towards Data Science. *The Concept of Artificial Neurons — Perceptrons in Neural Networks*. 2024. Disponível em: <https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc>.

ULTRALYTICS. *Object Detection with YOLO*. [S.l.], 2024. Acessado em: 3 fev. 2025. Disponível em: <https://docs.ultralytics.com/tasks/detect/>.

WOLFE, J. M. Visual search: How do we find what we are looking for? *Annual Review of Vision Science*, Annual Reviews, v. 1, p. 333–354, 2015.

XIE, S.; GIRSHICK, R.; DOLLÁR, P.; TU, Z.; HE, K. Aggregated residual transformations for deep neural networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 1492–1500, 2017.

YANG, L.; ZHENG, Z.; WANG, J.; SONG, S.; HUANG, G.; LI, F. Adadet: An adaptive object detection system based on early-exit neural networks. *IEEE Transactions on Cognitive and Developmental Systems*, IEEE, 2023.

YANG, W.; ZHAO, H.; LI, M.; SUN, C. Adadet: An adaptive framework for efficient object detection with early exits. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024. To appear.

ZAGORUYKO, S.; KOMODAKIS, N. Wide residual networks. In: *Proceedings of the British Machine Vision Conference (BMVC)*. [S.l.]: BMVA Press, 2016.

ZAIDI, S. S. A.; ANSARI, M. S.; ASLAM, A.; KANWAL, N.; ASGHAR, M.; LEE, B. A survey of modern deep learning based object detection models. *arXiv preprint arXiv:2104.11892*, 2021.