# Capítulo 3

# Uso de *Meta-Learning* em Tarefas de Aprendizado Profundo

Luis Gustavo Coutinho do Rêgo, Bárbara Stéphanie Neves Oliveira, Lucas Peres Gaspar, João Araújo Castelo Branco, José Antônio Fernandes de Macêdo

***Abstract***

*Deep Learning is a subarea of Machine Learning that uses neural networks with successive layers of data representation, which allow the performance of more complex tasks. Generally, these models produce better results when working with large volumes of data. However, in some situations, obtaining labeled data in large quantities for training these networks is not feasible. The meta-learning strategy aims to mitigate this problem, enabling learning models to learn quickly from other models initially trained for different tasks. This work introduces some meta-learning techniques, focusing on their use with Deep Learning models to solve tasks with fewer data.*

***Resumo***

*Aprendizado Profundo é uma subárea de Aprendizado de Máquina que utiliza redes neurais com sucessíveis camadas de representação dos dados, as quais permitem a realização de tarefas mais complexas. Em geral, esses modelos produzem melhores resultados quando trabalhados com grandes volumes de dados. Entretanto, em algumas situações, não é possível obter dados rotulados em grande quantidade para o treinamento dessas redes. A estratégia de meta-learning visa mitigar esse problema, fazendo com que modelos de aprendizagem consigam aprender, de forma rápida, a partir de outros modelos inicialmente treinados para diferentes tarefas. Este trabalho introduz algumas técnicas de meta-learning, focando em seu uso com modelos de Aprendizado Profundo para a resolução de tarefas com quantidade de dados reduzida.*

## 3.1. Introdução

Nos últimos anos, o Aprendizado Profundo, do inglês *Deep Learning* (também conhecido como Aprendizagem Profunda ou redes neurais profundas), impulsionou um rápido progresso em campos diversos como Visão Computacional e Processamento de Linguagem

Natural [Neves Oliveira et al. 2022a]. Dada a aplicabilidade desse aprendizado de representação em vários níveis por meio de muitas camadas de transformações, o Aprendizado Profundo substitui não apenas modelos superficiais de *pipelines* tradicionais de Aprendizado de Máquina, como também o processo trabalhoso da engenharia de *features*.

Grande parte do progresso recente do Aprendizado Profundo foi desencadeado por uma abundância de dados provenientes de sensores e aplicações da *Web*. No entanto, apesar de muitos avanços, ainda há desafios a serem resolvidos, como precisar de grandes quantidades de dados para obter bons desempenhos, além de conseguir assimilar novos padrões presentes nos dados de forma rápida e menos custosa [Huisman et al. 2021]. À medida que o uso de modelos de Aprendizado Profundo aumentou, as dificuldades para contornar essas limitações e treinar esses algoritmos originaram um aumento no interesse por estudos de *meta-learning*.

Abordagens de *meta-learning* [Thrun 1998] são usadas para melhorarem soluções de modelos de aprendizagem ao replicar a capacidade de adaptação da mente humana em diferentes tarefas. Um dos aspectos interessantes da inteligência humana é sua capacidade em adaptar-se rapidamente a novos problemas e padrões com apenas poucas informações. Por exemplo, uma pessoa que sabe pilotar um carro manual terá facilidade em aprender a pilotar um carro automático. Isto deve-se ao fato de que humanos conseguem utilizar suas experiências passadas em contextos similares para aprender a realizar algo novo [Lake et al. 2017].

O conceito chave da estratégia descrita é: assim como as pessoas, modelos de aprendizagem também devem conseguir aproveitar o conhecimento obtido de tarefas passadas [Schmidhuber 1987] para adequarem-se mais rapidamente a novas tarefas utilizando poucas informações (dados). Dessa forma, abordagens de *meta-learning* visam "aprender como modelos de aprendizagem aprendem", ao invés de apenas generalizarem os dados de uma tarefa específica, tal como as abordagens tradicionais operam.

Essa solução tem sido utilizada em diversos cenários e obtido bons resultados quando comparada às soluções convencionais. No cenário de sistemas de recomendação, técnicas de *meta-learning* podem mitigar o problema de *cold-start* de uma recomendação, uma vez que não se possuem informações prévias sobre um determinado usuário [Wang et al. 2022]. No contexto de trajetórias, a tarefa de planejamento de rotas em terrenos desconhecidos pode beneficiar-se dos conceitos de *meta-learning* dada a falta de informações e o baixo poder computacional de equipamentos embarcados contidos em veículos autônomos usados em experimentações [Visca et al. 2022].

Nos últimos anos, sugiram diversas estratégias para o uso de *meta-learning*. Uma forma de conseguir classificá-las é fazendo uma distinção sobre a metodologia de aprendizado que pode ser dividida em três tipos: abordagens baseadas em métricas, modelos e otimização. A Figura 3.1 mostra o desempenho de algumas dessas técnicas para a tarefa de classificação de imagens para o conjunto de dados *miniImageNet* [Vinyals et al. 2016].

O objetivo do presente trabalho é apresentar os conceitos básicos de *meta-learning* e como utilizar suas abordagens em tarefas de Aprendizado Profundo. Em resumo, este capítulo está organizado da seguinte forma: a Seção 3.2 apresenta a motivação por trás de alguns conceitos do Aprendizado Profundo e expõe as definições iniciais relacionadas ao

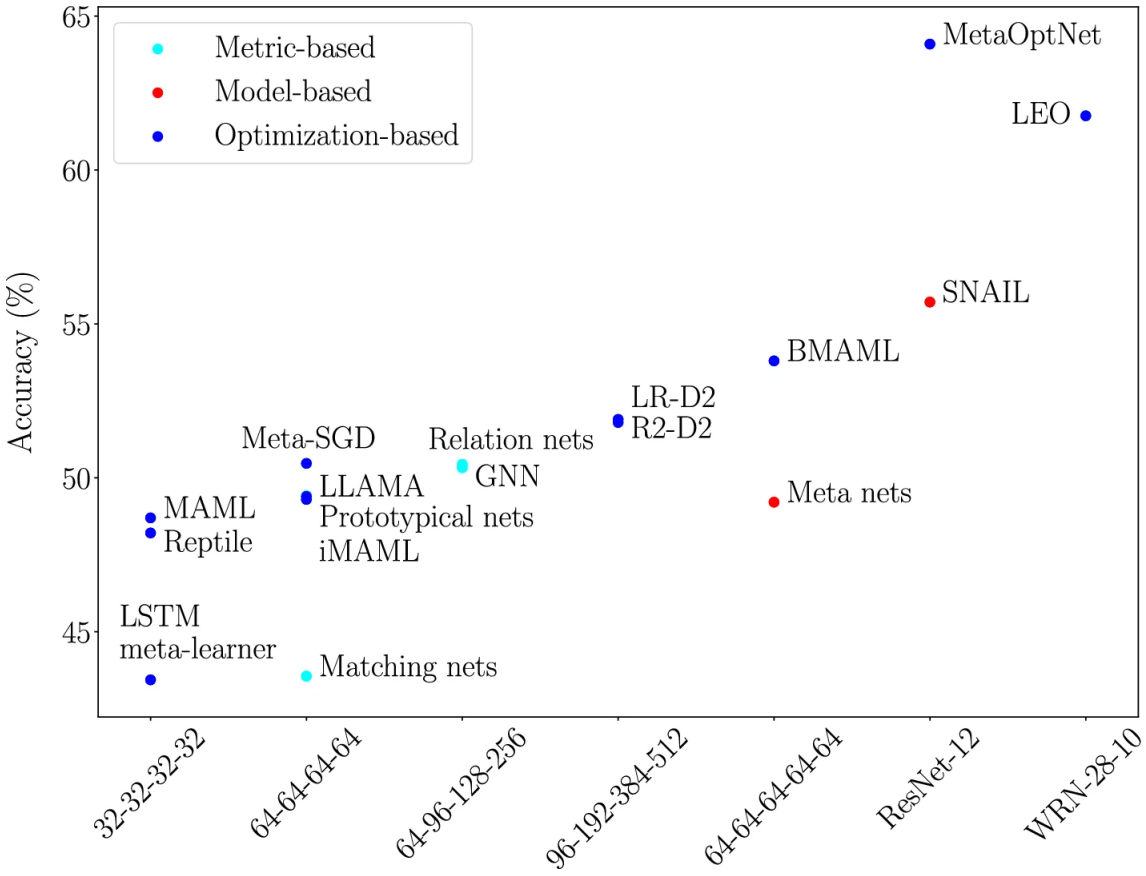


**Figura 3.1. Comparativo da acurácia (eixo $y$) entre as abordagens do *meta-learning* para o conjunto de dados miniImageNet utilizando diferentes redes como base (eixo $x$) [Huisman et al. 2021].**

campo de *meta-learning*. A Seção 3.3 mostra as diferentes abordagens de como resolver problemas com *meta-learning*. A Seção 3.4 apresenta um exemplo prático do uso de técnicas de *meta-learning* para a resolução de uma tarefa de classificação de imagens e, por fim, as Seções 3.5 e 3.6 apresentam algumas limitações, desafios de pesquisa e as considerações finais, respectivamente.

## 3.2. Fundamentação Teórica

Esta seção revisa os conceitos básicos sobre Aprendizado Supervisionado e os princípios relacionados à *meta-learning* necessários para o bom entendimento deste trabalho.

### 3.2.1. Aprendizado Supervisionado

A maneira usual de fazer com que uma máquina realize alguma tarefa envolve um(a) programador(a) e um *software* que contém regras a serem seguidas para transformar dados de entrada em respostas apropriadas. O Aprendizado de Máquina inverte esse processo: a máquina analisa os **dados de entrada** e as **respostas correspondentes** e descobre quais que devem ser as regras.

No Aprendizado de Máquina, essa definição trata-se de um tipo bastante espe-

cífico de aprendizagem chamada de **aprendizagem supervisionada**. Existem diferentes maneiras de construir algoritmos de Aprendizado de Máquina, cada uma com suas próprias vantagens e desvantagens. Assim, neste trabalho, apenas o aprendizado supervisionado, um dos tipos mais básicos, é referenciado para entendimento do conceito de *meta-learning*.

Tal como no Aprendizado de Máquina, o Aprendizado Profundo também possui diferentes tipos de aprendizagem. Para entender um pouco as diferenças de como o aprendizado supervisionado funciona entre modelos tradicionais e profundos, considere três fatores importantes:

1. **Dados de entrada.** Se a tarefa for identificar imagens de animais (tarefa comumente chamada de classificação de imagem), os dados de entrada serão as próprias imagens dos animais. A Figura 3.2 ilustra uma tarefa de classificação de imagens, em que os dados de entrada são imagens de formigas e joaninhas.

2. **Dados de saída que informam a que classes pertencem os dados de entrada.** Para a tarefa de classificação de imagens, são esperados valores como classes ou rótulos[1] que identificam as imagens de entrada em tipos específicos, como formiga e joaninha, dispostas em grupos com cores diferentes na Figura 3.2. A junção dos dados de entrada com os dados de saída em forma de pares (dados anotados) é comumente referenciada na literatura como conjunto de dados de treinamento (*training set*).

3. **Uma forma de medir se o modelo está realizando um bom trabalho.** Isso é necessário para determinar a distância entre o que o modelo julga ser uma certa imagem (se é uma formiga, por exemplo. O que o modelo julga ser correto é chamado de *classe predita*) e o que é realmente esperado (e se, na verdade, for uma joaninha? A informação correta sobre o que é uma imagem vinda do conjunto de dados é chamada de *classe verdadeira*). A medição é usada como um sinal de *feedback* para ajustar a maneira como o modelo funciona. Essa etapa de ajuste é o que é conhecido como aprendizado. Um modelo entende "como aprender" quando consegue generalizar o total de classes preditas com as classes verdadeiras.

Um sistema de aprendizado supervisionado é treinado em vez de explicitamente programado sendo apresentado a muitos dados e respostas relevantes para encontrar estrutura estatística que eventualmente permite que o sistema crie regras para automatizar a tarefa. Por exemplo, na Figura 3.2 a ilustração demonstra que o treinamento do modelo de classificação é realizado usando como entrada o conjunto de dados formado por muitas imagens de formigas e joaninhas. Em seguida, o conhecimento adquirido pelo modelo sobre as imagens é testado com uma nova imagem não vista anteriormente.

[1]Apesar de classes e rótulos servirem para o mesmo propósito, esses conceitos possuem algumas diferenças entre si: uma classe é uma categoria onde é possível agrupar instâncias dos dados (por exemplo, identificar quais imagens possuem apenas formigas, joaninhas, etc.), enquanto um rótulo é uma categoria que permite diferenciar os dados, ou seja, uma instância pode ser associada a um ou mais rótulos (por exemplo, identificar se uma imagem possui uma formiga e uma joaninha). Neste capítulo, os problemas apresentados lidam apenas com classes.

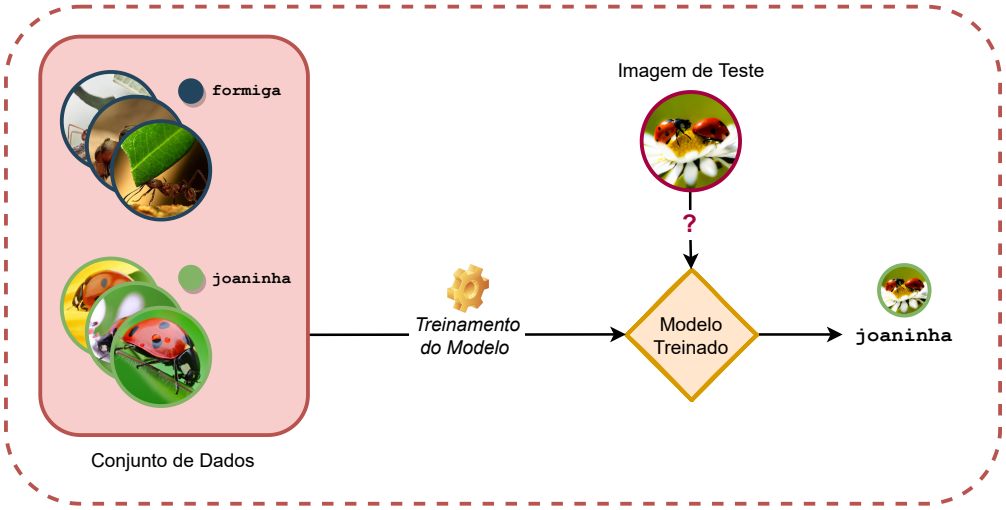


**Figura 3.2. Aprendizado supervisionado para o problema de classificação de imagens. O conjunto de dados formado por imagens pertencentes às classes `formiga` e `joaninha` é usado como entrada para um modelo de classificação. Após treinado, uma nova imagem que não pertence ao conjunto inicial usado para treinar o modelo (imagem de teste) é passada para o classificador treinado. O modelo realiza uma predição correta, informando que a imagem pertence à classe `joaninha`.**

O problema central em Aprendizado de Máquina e Aprendizado Profundo é transformar dados significativamente. Em outras palavras, aprender representações úteis dos dados de entrada disponíveis que aproximem das saídas esperadas. O Aprendizado Profundo oferece uma nova visão sobre o aprendizado de representações a partir do uso de camadas sucessivas de representações cada vez mais significativas por meio de redes neurais. Não só isso, mas o principal motivo que influencia na escolha desses tipos de modelos é o fato de exigirem pouca engenharia de *features* [Neves Oliveira et al. 2022b], podendo elencar automaticamente as características mais importantes presentes nos dados (como as linhas, texturas, antenas de uma formiga, patas de um gato, etc., em uma tarefa de classificação de imagens), conforme a arquitetura utilizada.

Uma prática será feita adiante com um modelo de rede neural profunda para ilustrar o funcionamento dessas camadas, mas como representações aprendidas podem parecer visualmente? Observe a Figura 3.3 que possui um exemplo de rede neural com várias camadas que transformam uma imagem de entrada de um dígito. As camadas da rede transformam a imagem do dígito em representações cada vez mais diferentes da imagem original e cada vez mais informativas sobre o resultado.

Na Seção 3.4, uma prática será feita com uma tarefa de classificação de imagens utilizando tanto Aprendizado Profundo convencional[2] quanto *meta-learning*. Nas duas abordagens, será usada como base um tipo de arquitetura de redes profundas, chamada de Redes Neurais Convolucionais (CNNs, de *Convolutional Neural Networks*), que começaram a obter ótimos resultados em competições de classificação de imagens entre os

[2]Neste trabalho, o termo "convencional" é usado para referenciar qualquer problema que não utilize conceitos de *meta-learning*. Por exemplo, tarefa de classificação *convencional* com Aprendizagem Profunda.

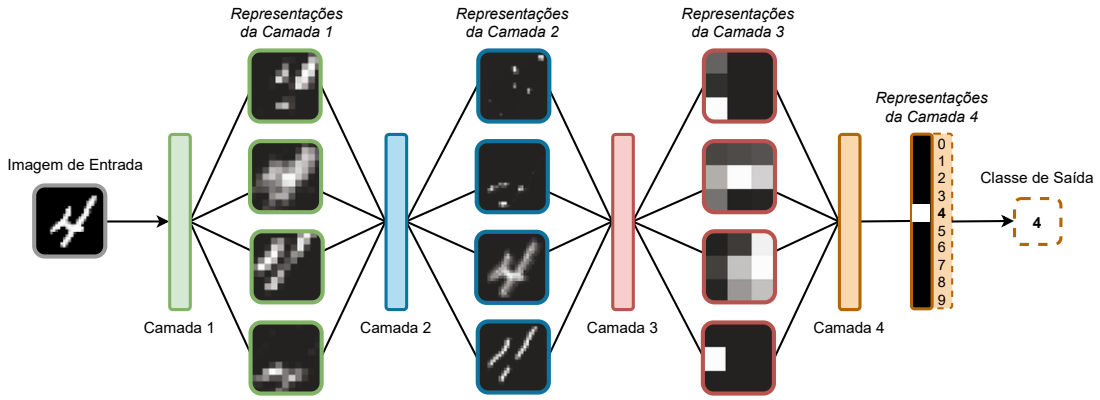


**Figura 3.3. Representações de dados aprendidas por um modelo de classificação de imagens de dígitos. Adaptado de [Chollet 2021].**

anos de 2011 e 2015. Mais especificamente, será usada a *ResNet* [He et al. 2016], uma modificação mais moderna das redes convolucionais.

Contudo, por que CNNs obtiveram bastante sucesso no problema de classificação de imagens? Para entender genericamente como CNNs funcionam, primeiro considere que o objetivo dessas arquiteturas é identificar algo, seja um ser vivo ou um objeto, em uma imagem. Desta forma, parece razoável afirmar que o modelo não deve preocupar-se com a localização precisa de um certo objeto ou ser na imagem. Idealmente, o modelo deve explorar um conhecimento extra sobre as imagens, isto é, precisa saber algumas das características principais que definem o que queria identificar.

Pelo fato das CNNs serem arquiteturas renomadas, a presença delas na literatura é bastante abrangente. O objetivo do restante desta subseção é apenas introduzir seus conceitos básicos intuitivamente e mostrar como usar uma CNN simples de forma prática. [Chollet 2021, Dumoulin and Visin 2016, Zhang et al. 2021] são alguns dos livros e artigos que possuem uma base sólida, da teoria à prática, sobre CNNs.

**Como Usar uma Arquitetura Simples Baseada em CNN.** A Figura 3.3 representa o funcionamento das camadas de um modelo de classificação de imagens de dígitos manuscritos. Será se é possível construir algo parecido (ou melhor) como o modelo ilustrado na Figura 3.3 com uma CNN?

O problema em questão consiste em classificar imagens de dígitos manuscritos em escala cinza ($28 \times 28$ *pixels*) em dez (10) classes compostas por números de zero (0) a nove (9). O conjunto de dados utilizado é o MNIST [3], um clássico da comunidade de Aprendizado de Máquina, que existe há quase tanto tempo quanto o próprio campo, composto por 60.000 imagens de treinamento e mais 10.000 imagens de teste. O Programa 3.1 mostra como carregar o conjunto de dados MNIST através da biblioteca *Keras*. `train_images` e `train_classes` formam o conjunto de treinamento, dados com os quais o modelo aprenderá, e o modelo será então testado com o conjunto de teste, formado por `test_images` e `test_classes`. As imagens são codificadas como matrizes

[3] `http://yann.lecun.com/exdb/mnist/`

*NumPy* e as classes são uma matriz de dígitos que variam de 0 a 9. As imagens e as classes têm uma correspondência de um para um e foram montadas pelo Instituto Nacional de Padrões e Tecnologia (o NIST no MNIST) na década de 1980.

```
import numpy as np
from tensorflow.keras.datasets import mnist

(train_images, train_classes), (test_images, test_classes) =
 ↪  mnist.load_data()

print(train_images.shape, train_classes.shape)
# ((60000, 28, 28), (60000,))

print(test_images.shape, test_classes.shape)
# ((10000, 28, 28), (10000,))

print("Classes do MNIST =
 ↪  {}".format(np.unique(train_classes.tolist() +
 ↪  test_classes.tolist())))
# Classes do MNIST = [0 1 2 3 4 5 6 7 8 9]
```

**Programa 3.1. Carregando o conjunto de dados MNIST com o *Keras*.**

O fluxo a partir desse momento será o seguinte: primeiro, a CNN será alimentada com os dados de treinamento, `train_images` e `train_classes`. A rede então aprenderá a associar imagens e classes. Por fim, será solicitado à rede que produza predições para `test_images` e será verificado se essas predições correspondem às classes de `test_labels` (ao invés de testar para apenas uma imagem, como é demonstrado na Figura 3.3). O Programa 3.2 constrói uma arquitetura simples baseada em camadas CNN, formada apenas com camadas `Conv2D` (convolução) e `MaxPooling2D` (*pooling*).

Observe que no Programa 3.2 o modelo da CNN está configurado para processar entradas de tamanho `(28, 28, 1)` (altura da imagem, largura da imagem e canais da imagem, respectivamente)[4], que é o formado das imagens do MNIST. É possível ver no código o formato das saídas de cada camada: para a maioria delas, um tensor de tamanho três (`kernel_size=3`) na forma `(altura, largura, canais)`. As dimensões de largura e altura tendem a diminuir à medida que se passa pelas camadas. O número de canais é controlado pelo primeiro argumento `filters` passado nas camadas de convolução (filtros de tamanho 32, 64 ou 128).

Após a última camada de convolução, a saída possui a forma `(3, 3, 128)`, que consiste em um mapa de *features* $3 \times 3$ com 128 filtros. O próximo passo é usar essa saída como entrada em um classificador densamente conectado (`Dense`), que processa vetores

[4]As camadas de convolução operam sobre tensores de nível três (3) chamados de mapas de *features*, que possuem dois eixos espaciais (altura e largura), bem como um eixo de profundidade (também chamado de canal). Para uma imagem RGB, a dimensão do eixo de profundidade é 3, pois a imagem possui três canais de cores: vermelho, verde e azul. Para uma imagem em preto e branco, como os dígitos MNIST, a profundidade é 1 (níveis de cinza). Além disso, canais podem ser usados para representar filtros [Chollet 2021].

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(28, 28, 1))

# Camada/bloco 1
conv32 = layers.Conv2D(filters=32, kernel_size=3,
 ↪  activation="relu")(inputs) # Saída (26, 26, 32)
pool1 = layers.MaxPooling2D(pool_size=2)(conv32) # Saída (13, 13,
 ↪  32)

# Camada/bloco 2
conv64 = layers.Conv2D(filters=64, kernel_size=3,
 ↪  activation="relu")(pool1) # Saída (11, 11, 64)
pool2 = layers.MaxPooling2D(pool_size=2)(conv64) # Saída (5, 5,
 ↪  64)

# Camada/bloco 3
conv128 = layers.Conv2D(filters=128, kernel_size=3,
 ↪  activation="relu")(pool2) # Saída ( 3, 3, 128)
flatten = layers.Flatten()(conv128) # Saída (1152,)

# Camada 4
outputs = layers.Dense(10, activation="softmax")(flatten) # Saída
 ↪  (10,)
cnn_model = keras.Model(inputs=inputs, outputs=outputs)
```

**Programa 3.2. Inicializando uma CNN simples.**

de 1 dimensão (1D). Como a saída atual é um tensor de 3 dimensões, a camada `Flatten` é usada para converter 3D para 1D antes de passar para a camada `Dense`, que possui 10 saídas possíveis e uma função de ativação *softmax*[5].

O Programa 3.3 treina a CNN criada. Antes é feito um pré-processamento para padronizar os dados na forma que o modelo espera e dimensioná-los para que todos os valores estejam no intervalo $[0, 1]$. Como a saída do modelo pode conter 10 diferentes classes, é usada a função de perda (ou *loss*) *categorical crossentropy*, em especial, a versão esparsa `sparse_categorical_crossentropy`, já que as classes tratam-se de números inteiros. A CNN obtém uma acurácia de aproximadamente 99% sobre o conjunto de teste. Note que o modelo criado possui a mesma quantidade de camadas que o da Figura 3.3.

### 3.2.2. *Meta-Learning*

Modelos treinados utilizando arquiteturas de Aprendizado Profundo, em geral, necessitam de uma grande quantidade de dados de entrada para obterem bons resultados. Seres

[5]A função *softmax* é usada como função de ativação na camada de saída em modelos que predizem uma distribuição de probabilidade multinomial, como para os que lidam com mais de duas classes.

```
# Padronização dos dados
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype("float32") / 255

# Treinamento do modelo
cnn_model.compile(optimizer="rmsprop",
                  loss="sparse_categorical_crossentropy",
                  metrics=["accuracy"])
cnn_model.fit(train_images, train_classes, epochs=5,
 ↪  batch_size=64) # São usadas 5 épocas para treinamento

# Teste do modelo
test_loss, test_acc = cnn_model.evaluate(test_images,
 ↪  test_classes)
print("Acurácia do teste = {:.3f}".format(test_acc))
# Acurácia do teste = 0.991
```

**Programa 3.3. Treinando e avaliando a CNN com o conjunto de dados de imagens MNIST.**

humanos, no entanto, realizam tarefas completamente novas com apenas poucos exemplos ou conseguem aprender uma nova habilidade com poucas instruções caso possuam conhecimento prévio de uma habilidade similar. Por exemplo, uma pessoa A consegue identificar outra pessoa B em uma fotografia tendo observado poucas imagens da pessoa B previamente; ou, um aluno que sabe tocar violão precisará de pouca ajuda para aprender a tocar uma guitarra.

Para tentar simular o aprendizado rápido de novas tarefas baseado em conhecimentos prévios de outras tarefas distintas, modelos de *meta-learning* foram desenvolvidos para tentar mimetizar o aprendizado humano. Esses modelos são treinados utilizando não um conjunto de dados convencionais, como imagens e suas classes em um problema de classificação de imagens, mas sim tarefas inteiras treinadas em conjuntos de dados diferentes, como classificadores de imagens de carros, classificadores de motos e classificadores de bicicletas. Modelos treinados com base em um conjunto de tarefas poderão se adaptar a novas tarefas similares, como classificação de imagens de ônibus, uma vez que já possuem conhecimento prévio de tarefas similares. Por conta disso, modelos de *meta-learning* também são conhecidos como modelos capazes de "aprender a aprender".

A Figura 3.2 exemplifica como ocorre, de forma geral, o treinamento de um modelo convencional de classificação de imagens de dois tipos específicos de insetos. Nesse cenário, imagens das duas classes são fornecidas como entrada para a arquitetura e um modelo é gerado, podendo categorizar novas imagens que lhes forem fornecidas. A Figura 3.4 exemplifica como acontece o treinamento de um meta-modelo: em um primeiro momento, tarefas distintas de classificação de animais (Tarefas $T_1$, $T_2$ e $T_3$) são treinadas isoladamente (fase de aprendizado base, ou *base learning*, em inglês) e os resultados da soma das suas funções de perda são utilizadas como resultado da função de perda no

treinamento do meta-modelo (fase de meta-aprendizado, ou *meta-learning*, em inglês). Uma vez que o meta-modelo tenha sido treinado, ou seja, que tenha aprendido sobre o conhecimento gerado pelos modelos das tarefas anteriores, pode-se realizar o processo de *fine-tuning* em uma nova tarefa ($T_4$) para que o meta-modelo seja apto a classificar os novos tipos de animais da nova tarefa.

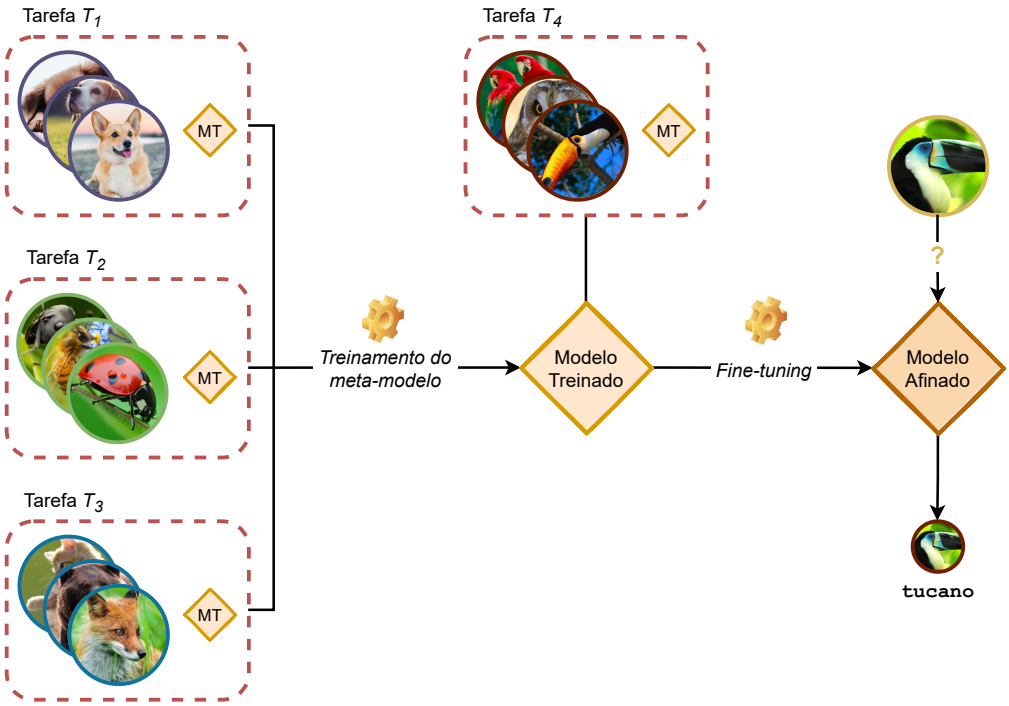


**Figura 3.4. Treinamento de um meta-modelo utilizando diferentes tarefas e posterior *fine-tuning* com uma tarefa inédita. A sigla MT representa o modelo convencional treinado para cada tarefa individualmente. O esquema mostra que o meta-modelo foi treinado com diferentes tarefas de classificação de animais: raças de cães (Tarefa $T_1$), insetos (Tarefa $T_2$) e mamíferos no geral (Tarefa $T_3$). O meta-modelo, que até então possui um conhecimento prévio das Tarefas $T_1$, $T_2$ e $T_3$, é treinado novamente com uma Tarefa $T_4$ que classifica tipos de aves. Por fim, o modelo é testado com uma nova imagem que antes do *fine-tuning* só poderia ser predita pela Tarefa $T_4$.**

#### 3.2.2.1. Definição

O processo de *meta-learning* utiliza um conjunto de tarefas, extraído de um grupo maior de tarefas semelhantes entre si, e consiste em duas fases principais [Upadhyay et al. 2021]:

1. **Meta-treinamento.** Nesta fase, o treinamento ocorre em dois níveis distintos: (i) o aprendizado interno, que treinará meta-modelos para cada uma das tarefas utilizadas como entrada; e (ii) o aprendizado externo, que treinará o meta-modelo baseado nos valores das funções de perda das tarefas do nível de aprendizado interno. O mo-

delo final gerado na fase de meta-treinamento consegue generalizar o aprendizado realizado em todas as tarefas de entrada fornecidas.

2. **Meta-teste.** Também conhecida por fase de *fine-tuning*, a fase de meta-teste utiliza o modelo treinado na fase anterior como conhecimento prévio para adaptá-lo a uma nova tarefa não vista anteriormente.

#### 3.2.2.2. Organização de Tarefas para *Meta-Learning*

Uma das formas mais comuns de modelar tarefas utilizadas como entrada para o treinamento de meta-modelos é através da divisão *N-way*, *K-shot*, que funciona da seguinte forma: cada tarefa deverá possuir $N$ classes distintas e $K$ exemplos para cada classe em seu conjunto de dados de treinamento (também chamado de conjunto de suporte ou *support set*). A Figura 3.5 exemplifica algumas divisões *N-way*, *K-shot* do conjunto de dados de treinamento para tarefas de classificação de imagens. Para o conjunto de dados de teste da tarefa (também chamado de conjunto de consulta ou *query set*), nenhuma restrição é estabelecida.

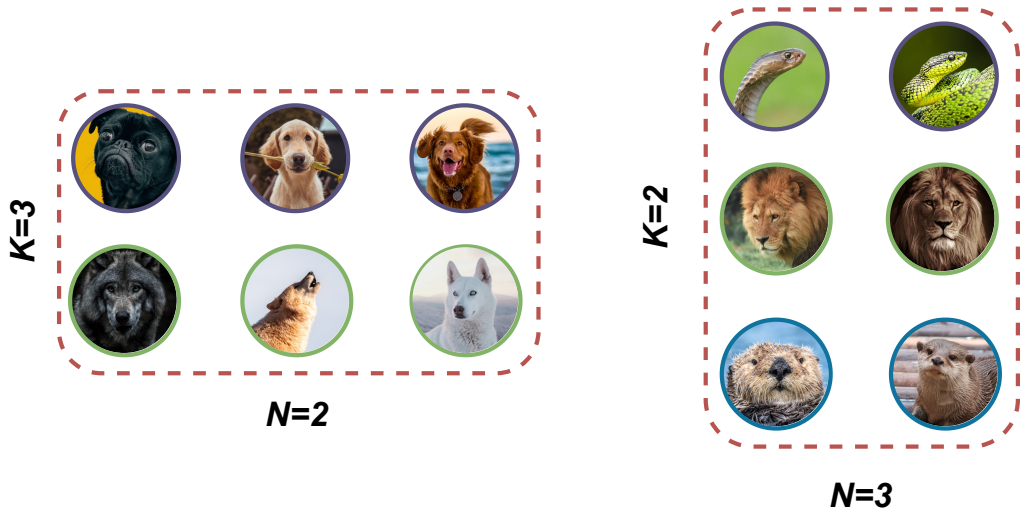


**Figura 3.5. Diferentes exemplos de conjuntos de dados *N-way*, *K-shot* para modelagem de tarefas de meta-modelos.**

Para o treinamento e teste de um meta-modelo, ou seja, um meta-treinamento e um meta-teste, respectivamente, exemplos de tarefas (também chamadas de episódios) são usadas com uma configuração *N-way*, *K-shot*. A Figura 3.6 ilustra como ocorre a divisão de tarefas no treinamento de um meta-modelo.

Em domínios mais específicos, a quantidade de dados disponível não é suficiente para o treinamento de modelos de Aprendizado Profundo com *meta-learning*. Em geral, esses conjuntos de dados possuem um subconjunto rotulado manualmente por algum especialista de domínio. Ou seja, ao mapear esse cenário para uma tarefa com configuração *N-way*, *K-shot*, têm-se valores de $N$ e $K$ muito pequenos. Tarefas desse tipo são chamadas de problemas de *few-shot learning* (ou aprendizagem com poucos exemplos, em tradução livre).

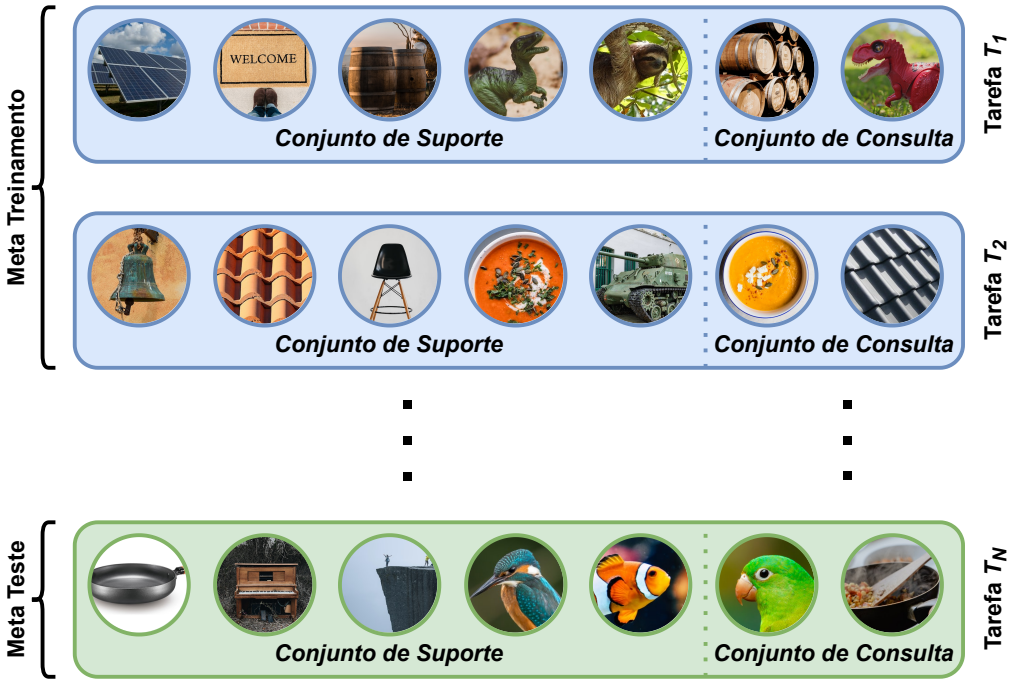


**Figura 3.6. Tarefa de classificação $N$-*way*, $K$-*shot*, onde $N = 5$ e $K = 1$. Adaptado de [Huisman et al. 2021].**

## 3.3. Técnicas de *Meta-Learning* Aplicadas a Aprendizado Profundo

Uma vez que a estrutura de tarefas é definida, existem algumas técnicas que podem ser utilizadas para o treinamento do meta-modelo. Essas diferentes abordagens variam com base em como o meta-modelo se adapta perante às tarefas do treinamento. Essas abordagens podem ser classificadas em três categorias, baseadas em:

1. **Métricas.** A abordagem baseada em métricas geralmente utiliza técnicas não paramétricas (por exemplo, $k$-vizinhos mais próximos) para generalizar as tarefas utilizadas para treinamento do meta-modelo.

2. **Modelos.** Também conhecida como abordagem "caixa preta", seus métodos treinam uma rede neural inteira, com alguns dados de treinamento no conjunto de suporte e um conjunto inicial de meta-parâmetros. Depois, guardam uma representação interna e fazem as predições no conjunto de consultas.

3. **Otimização.** A abordagem baseada em otimização trata a parte de adaptação do processo como um problema de otimização e utiliza algoritmos tradicionais para a atualização dos parâmetros do meta-modelo, como o gradiente descendente.

Nesta seção, cada uma das abordagens descritas brevemente acima é apresentada em detalhe, como também alguns dos algoritmos mais utilizados.

### 3.3.1. Abordagem Baseada em Métricas

A primeira abordagem baseada em métricas visa encontrar um **espaço vetorial** que possa representar bem as características dos dados de diversas tarefas distintas. Assim, novas tarefas que possuem uma pequena quantidade de exemplos podem ser aprendidas apenas comparando o novo conjunto de dados com as representações vetoriais previamente obtidas, das quais se sabem as classes. Quanto maior for a semelhança entre um novo dado representado e uma representação presente no espaço vetorial, maior é a chance de que o dado tenha a mesma classe que a da representação.

Formalmente, a ideia é encontrar o meta-conhecimento $\omega$ na forma de um modelo com parâmetros $\theta$ (a qual será referida como $f_\theta : \mathrm{R}^N \rightarrow \mathrm{R}^M$). Dessa forma, é possível criar um *kernel de similaridade*, permitindo que duas representações vetoriais $\mathbf{x}_1$ e $\mathbf{x}_2$ possam ser comparadas, onde representações mais similares pertencem a um mesmo grupo. Logo, a predição de novos dados dá-se comparando novas representações $\mathbf{x}$ com representações vetoriais $\mathbf{x}_i$ já conhecidas.

Na Figura 3.7, as retas representam $f_\theta(\mathbf{x})$, onde $\mathbf{x}$ é uma representação. Os dados em vermelho representam dados cujos rótulos são conhecidos (conjunto de suporte), denotado por $S$, enquanto o dado em azul representa um novo dado. A partir dessa projeção, é possível identificar a qual elemento de suporte o novo dado mais se assemelha, permitindo então sua classificação.

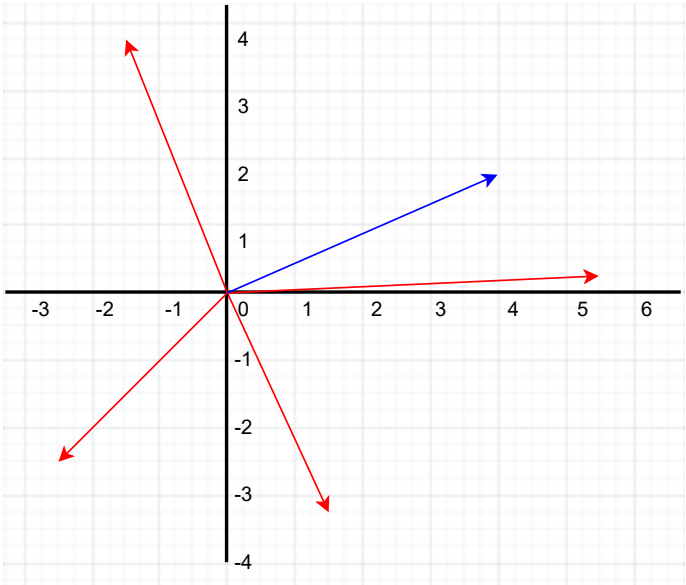


**Figura 3.7. Ilustração dos métodos baseados em métricas. O vetor azul indica a representação vetorial de um novo dado, enquanto os vermelhos, dados conhecidos.**

Nos métodos baseados em métricas, o aprendizado é tido como **não paramétrico**: o modelo $f_\theta$ não sofre nenhum tipo de modificação quando lida com novas tarefas, pois ele é o meta-conhecimento aprendido pelo método. Os dados apenas são projetados e comparados. Dada uma tarefa $T = (\mathrm{D}_T^{tr}, \mathrm{D}_T^{te})$, uma nova representação vetorial $\mathbf{x} \in \mathrm{D}_T^{te}$ e uma medida de similaridade $s_\theta = s(f_\theta(\mathbf{x}_i), f_\theta(\mathbf{x}_j))$, a distribuição probabilística de $\mathbf{x}$ sobre as classes $Y$ existentes pode ser encontrada pela Equação 1, onde $y_i$ é um vetor *one-*

*hot* (vetor de zeros com o valor um na posição da respectiva classe). A partir do resultado da equação, a classe predita será aquela com maior valor.

$$p_\theta(Y|\mathbf{x}, \mathrm{D}^{tr}) = \sum_{(\mathbf{x}_i, y_i) \in \mathrm{D}^{tr}} s_\theta(\mathbf{x}, \mathbf{x}_i) y_i \tag{1}$$

As principais vantagens do uso de técnicas baseadas em métricas são que (i) a ideia de realizar predições baseadas em similaridade é conceitualmente simples e (ii) elas podem ser rápidas no teste quando as tarefas são pequenas, pois as redes não precisam fazer ajustes específicos na arquitetura [Huisman et al. 2021]. No entanto, quando as tarefas de meta-teste tornam-se mais distantes das tarefas usadas no meta-treinamento, essas técnicas são incapazes de entender novas informações específicas das tarefas. Além disso, conforme as tarefas aumentam, as comparações de pares podem se tornar computacionalmente mais caras. Por fim, a maioria das técnicas baseadas em métricas depende da presença de dados anotados, o que as tornam inaplicáveis para configurações fora do aprendizado supervisionado.

Existem diversos métodos baseados em métricas: redes siamesas [Koch et al. 2015], *matching networks* [Vinyals et al. 2016], *Graph Neural Network* [Garcia and Bruna 2017], dentre outras. Este trabalho focará apenas na *prototypical networks* [Snell et al. 2017].

#### 3.3.1.1. *Prototypical Networks*

O método baseia-se na ideia de que é mais fácil realizar classificações quando as classes estão representadas no mesmo espaço vetorial dos dados. Dado que $f_\theta$ realiza uma projeção $M$-dimensional, o objetivo é encontrar um ponto $\mathbf{c}_k$ (conhecido como protótipo ou *prototype*) para representar cada classe. Uma forma intuitiva de representar o protótipo $\mathbf{c}_k$ é encontrando o centro geométrico dos elementos que pertencem àquela classe, conforme representado na Figura 3.8. Comumente, os modelos $f_\theta$ são redes neurais, por isso o nome *prototypical networks*.

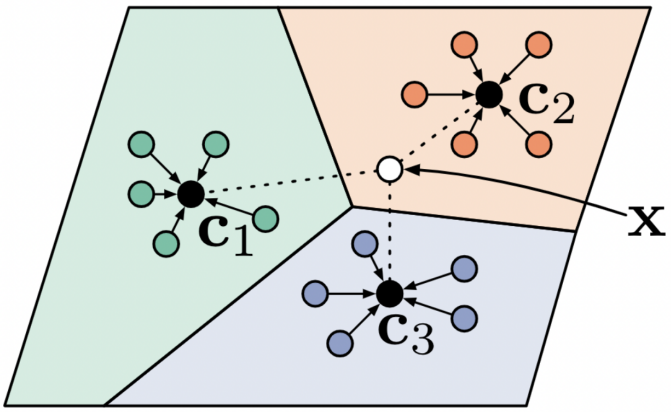


**Figura 3.8. Exemplo da técnica *prototypical networks*. As superfícies coloridas representam os dados de suporte projetados no espaço $\mathrm{R}^M$ utilizando $f_\theta$. Os pontos pretos representam os protótipos de cada classe. Um novo dado x é classificado conforme o protótipo mais próximo. Fonte: [Snell et al. 2017].**

Os protótipos podem ser calculados conforme a Equação 2. Para cada representação no conjunto de suporte de cada classe, o protótipo é caracterizado pelo seu centroide e $S_k$ representa o subconjunto dos exemplos do conjunto de suporte $S$ associados à classe $k$.

$$\mathbf{c}_k = \frac{1}{|S_k|} \sum_{(\mathbf{x}_i, y_i) \in S_k} f_\theta(\mathbf{x}_i) \tag{2}$$

A implementação do cálculo dos protótipos encontra-se no Programa 3.4. Inicialmente, as representações são projetadas no espaço vetorial utilizando o meta-modelo $f_\theta$. Em seguida, é criada uma matriz para representar todos os centroides, inicializadas com zeros (0's). Para cada classe, são selecionadas as representações projetadas que pertencem aquela classe, sendo tirada sua média. A matriz dos centroides é, por fim, retornada.

```
def compute_prototypes(model,X,y):
    projection = model.predict(X, verbose=0)
    centroids = np.zeros((y.shape[1], projection.shape[1]))
    for i in range(y.shape[1]):
        # Obtém a representação para a classe i
        idx = y.argmax(axis=1) == i
        # Computa a média
        centroids[i,:] = projection[idx].mean(axis=0)

    return centroids
```

**Programa 3.4. Cálculo dos protótipos de um conjunto de representações X e classes y, utilizando um modelo para projetar os dados.**

A partir dos protótipos encontrados e de uma medida de distância $d : \mathrm{R}^N \times \mathrm{R}^M \rightarrow [0, +\infty)$, a probabilidade de um novo ponto $\mathbf{x}$ pertencer a cada classe conhecida dá-se pela aplicação da função exponencial normalizada (ou *softmax*) sobre o negativo da distância de $\mathbf{x}$ para o protótipo $\mathbf{c}_k$ de cada classe, conforme apresentado na Equação 3. Consequentemente, a classe predita será a de maior probabilidade.

$$p_\theta(y = k|\mathbf{x}) = \frac{\exp(-d(f_\theta(\mathbf{x}), \mathbf{c}_k))}{\sum_{k'} \exp(-d(f_\theta(\mathbf{x}), \mathbf{c}_{k'}))} \tag{3}$$

O Programa 3.5 apresenta o cálculo da probabilidade presente na Equação 3. Nele, é possível informar um conjunto inteiro de representações a serem classificadas. Isso é possível, pois a biblioteca *NumPy* permite a organização de matrizes para realizar múltiplas operações simultaneamente. Nesse caso, a matriz dos protótipos é $K \times M$, enquanto a matriz das projeções dos dados de consulta é $Nq \times M$, onde $Nq$ é a quantidade de representações. Os *reshapes* realizados no código permitem que cada uma das $Nq$ representações seja operada com cada um dos $K$ protótipos. Ao tirar a norma, obtém-se um resultado de $Nq \times K \times 1$, que indica a distância euclidiana entre cada representação e protótipo. Por fim, é possível aplicar o *softmax* para cada linha da matriz negativa das distâncias

(transformando ela em uma medida de similaridade), resultando na probabilidade das representações.

```
def predict_proba(model, prototypes, query):
    projections = model(query)

    projections = tf.reshape(projections,
     ↪  (projections.shape[0],1,projections.shape[1],1))
    prototypes =
     ↪  prototypes.reshape(1,prototypes.shape[0],prototypes.shape[1],1)

    distances = tf.norm(projections - prototypes, axis=2, ord=np.inf)
    distances = tf.reshape(distances, (len(query), len(prototypes)))

    probabilities = keras.activations.softmax(-distances, axis=1)
    return probabilities
```

**Programa 3.5. Cálculo das probabilidades de cada elemento do conjunto de consulta pertencer a cada classe a partir de seus protótipos e um modelo para projetar os dados.**

O processo do aprendizado do meta-modelo dá-se por minimizar a log-probabilidade negativa $J(\theta) = -\log p_\theta(y = k|\mathbf{x})$ da classe $k$ esperada, utilizando o gradiente descendente estocástico (ou algum outro método de gradiente). O Programa 3.6 demonstra uma implementação desse aprendizado.

```
def task_episode(model,
             support, support_labels,
             query, query_labels, optimizer):

    prototypes = compute_prototypes(model, support, support_label)
    loss = None
    with tf.GradientTape() as tape:
        probabilities = predict_proba(model, prototypes, query)
        right_probabilities =
         ↪  tf.reduce_sum(probabilities*query_labels, axis=1)
        neg_log_right_prob = -tf.math.log(right_probabilities)
        loss = tf.reduce_mean(neg_log_right_prob)

    gradient = tape.gradient(loss,model.trainable_variables)
    optimizer.apply_gradients(
        zip(gradient, model.trainable_variables))
```

**Programa 3.6. Implementação do processo de aprendizado de uma tarefa utilizando *prototypical networks*.**

### 3.3.2. Abordagem Baseada em Modelos

Enquanto a abordagem baseada em métricas visa encontrar um espaço vetorial que permita comparar bem as características dos dados, a abordagem baseada em modelos tenta fazer com que, durante o meta-treinamento, as tarefas mudem de alguma forma a **estrutura do modelo** manipulado internamente. Como o meta-modelo em questão possui um estado interno e uma representação da tarefa, quando apresentado a uma nova tarefa, ele deve conseguir capturar informações específicas das tarefas consideradas relevantes para usá-las em seguida.

Vale ressaltar que, como o próprio meta-modelo realiza operações em seu estado interno, eles são considerados "caixa-preta", reduzindo sua explicabilidade para os resultados obtidos. Além disso, como o meta-modelo deve reter informações de tarefas específicas e passadas, é necessário utilizar um mecanismo de memória, que pode ser interno ou externo.

As vantagens de usar a abordagem baseada em modelos inclui a flexibilidade da dinâmica interna dos sistemas e sua aplicabilidade mais ampla em comparação com a maioria das técnicas baseadas em métricas. No entanto, métodos baseados em modelo são frequentemente superados pelos baseados em métricas, podem não ter um bom desempenho quando apresentados a conjuntos de dados maiores, e generalizam menos para tarefas mais discrepantes entre si do que técnicas baseadas em otimização [Huisman et al. 2021].

Existem diversos algoritmos de *meta-learning* baseados em modelos, como o *Simple Neural Attentive Learner* (SNAIL) [Mishra et al. 2017], que propõe uma arquitetura de rede neural para agir como mecanismo de memória e a *Conditional Neural Processes* (CNP) [Garnelo et al. 2018] que não necessita de um módulo de memória. Esta seção irá focar na *Meta-Networks* [Munkhdalai and Yu 2017] e na abstração utilizada para seu funcionamento.

#### 3.3.2.1. *Meta-Networks*

O método consiste em utilizar redes neurais tanto para aprender uma nova tarefa quanto para generalizar seus conceitos para novas tarefas. *Meta-Networks* são formadas por duas redes neurais, como mostra a Figura 3.9: uma atuando no espaço vetorial proveniente das tarefas (generalizando um novo conceito) e a outra no espaço da tarefa (atividade para extração de *features* dos dados da tarefa). Essas redes possuem o nome de *meta-learner* e *base-learner*, respectivamente.

O fluxo de aprendizagem inicia-se no *base-learner*, o qual executará seu aprendizado sobre a tarefa, enviando um *feedback* para o *meta-learner* na forma de um relatório, contendo meta-informações sobre o aprendizado para as tarefas executadas nele.

Observe que o *feedback* enviado do *base-learner* para o *meta-learner* tem o intuito de mostrar o quão bem o *base-learner* está aprendendo sobre a tarefa. Ou seja, poderá ser passado o valor da função de perda como meta-informação para quantificar isso, assim como métricas mais adequadas para o domínio da aplicação.

Em seguida, o *meta-learner*, de posse do *feedback*, deverá aprender um mapea-

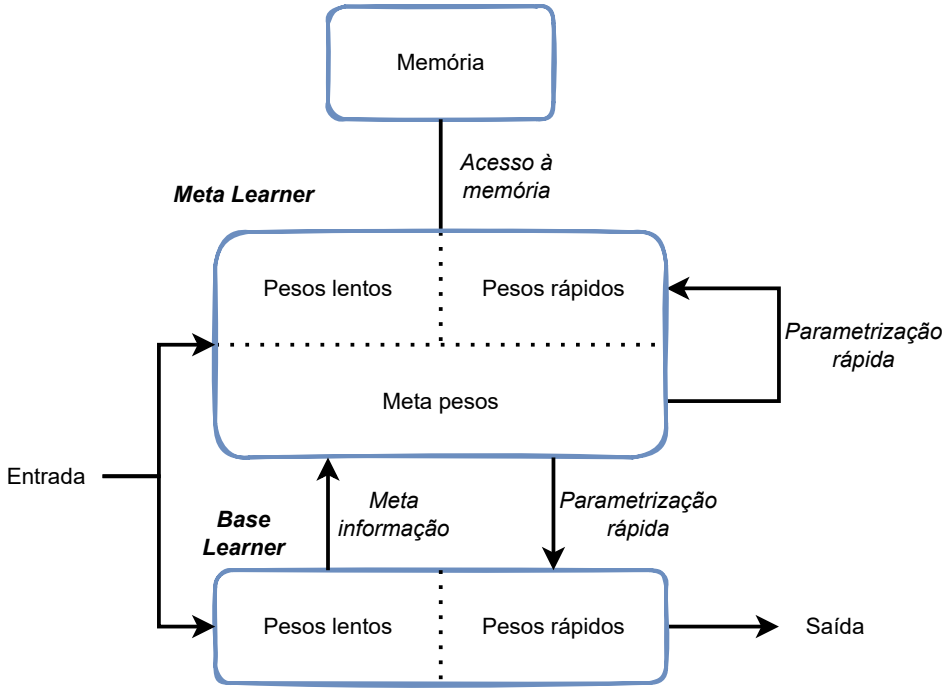


**Figura 3.9. Arquitetura de uma *Meta-Network*.**

mento desta informação à sua camada de pesos rápidos. Por fim, uma vez que sua camada de pesos provenientes da matriz de memória é atualizada, ele gerará uma representação de vetores indexada por cada tarefa vista para, então, realizar o treinamento sobre ela.

Como ele está operando no espaço vetorial, o *meta-learner* suporta aprendizado contínuo sobre tarefas agnósticas, atualizando seus próprios parâmetros e os do *base-learner* utilizado[6].

As seções a seguir apresentam mais detalhes sobre os componentes descritos anteriormente.

#### 3.3.2.2. *Meta-Learner*

Este componente é composto por uma função de representação vetorial e duas funções de geração rápida de pesos. Considere $m$ e $d$ como as funções de geração rápida de pesos e $u$ a função de representação vetorial. A função $m$ é responsável por aprender o mapeamento do *feedback* fornecido pelo *base-learner* à camada de pesos rápidos.

Além disso, a função $m$ deverá armazenar os pesos obtidos na matriz de memória $M$, indexada pela representação vetorial de tarefas fornecida pela função $u$. A função de aprendizagem $u$ é parametrizada pela camada de pesos lentos em conjunto com a camada de pesos rápidos. Desta forma, ela consegue capturar as funções de perda de cada tarefa passada.

[6]Para mais detalhes sobre a implementação de uma *Meta-Network*, ver o repositório do seu criador [Munkhdalai and Yu 2017]

Finalmente, após os gradientes serem atualizados na matriz de memória *M*, é realizado uma espécie de "sorteio" de *T* tarefas para compor o treino da função de aprendizado *d*. Essa etapa é muito importante, pois, como o treino ocorre sequencialmente, a ordem das tarefas passadas pode influenciar nos resultados obtidos.

A rede representada por *d* precisa aceitar um tamanho de entrada variável. Para tal, uma MLP pode ser usada para sumarizar os gradientes, entretanto, os experimentos realizados por [Munkhdalai and Yu 2017] demonstraram uma fraca convergência da *Meta-Network* quando configurada desta forma, sendo preferível utilizar uma rede *Long Short-Term Memory* (LSTM) [Hochreiter and Schmidhuber 1997] para esta atividade.

#### 3.3.2.3. *Base-Learner*

Diferentemente do *meta-learner*, o *base-learner*, denotado por *b*, possui uma única função de aprendizado, cujo objetivo consiste em aprender e extrair atributos da tarefa. Entretanto, ao contrário de redes neurais convencionais, essa é parametrizada tanto pela camada de gradientes rápidos quanto pela camada de gradientes lentos.

O interessante desse componente de aprendizado é que seus parâmetros são atualizados em momentos diferentes. Os gradientes lentos são atualizados durante o treinamento de *b*, enquanto os gradientes rápidos são atualizados pelo *meta-learner* a cada entrada da rede. A Figura 3.9 ilustra, de uma perspectiva mais geral, como ocorre essa troca de informações entre os componentes da *Meta-Network*.

Segundo [Munkhdalai and Yu 2017], é possível instanciar um *base-learner* mais simples, com apenas uma das camadas de gradiente, seja ela a rápida ou lenta, porém, durante seus experimentos, concluiu-se que são necessárias as duas camadas de gradientes para a *Meta-Network* convergir devidamente.

A Figura 3.10 mostra como uma MLP pode ser instanciada para ser utilizada em uma *Meta-Network*. Uma CNN modificada para atender os dois gradientes também pode ser utilizada e, segundo [Munkhdalai and Yu 2017], a CNN apresentou melhores resultados que a MLP proposta inicialmente para uso em *Meta-Networks*.

### 3.3.3. Abordagem Baseada em Otimização

Modelos de Aprendizado Profundo convencionais atualizam seus parâmetros utilizando o algoritmo de retro-propagação de seus gradientes. Entretanto, em muitas aplicações, são necessárias grandes quantidades de dados para esse treinamento. As técnicas de *meta-learning* baseadas em otimização focam em **otimizar meta-parâmetros** para que novas tarefas sejam aprendidas rapidamente, mas com uma pequena quantidade de dados.

A maioria das técnicas utiliza otimizações de dois níveis: a primeira otimização, também chamada de otimização interna, otimiza os parâmetros das tarefas (utilizando o gradiente descendente, por exemplo) que serão utilizadas para otimizar um meta-modelo, como em um modelo convencional; já a segunda otimização, também chamada de otimização externa, utiliza o resultado das funções de custo das tarefas de entrada e generaliza o aprendizado das tarefas utilizadas no otimizador interno.

Dentre as diversas técnicas de *meta-learning* disponíveis baseadas em otimização

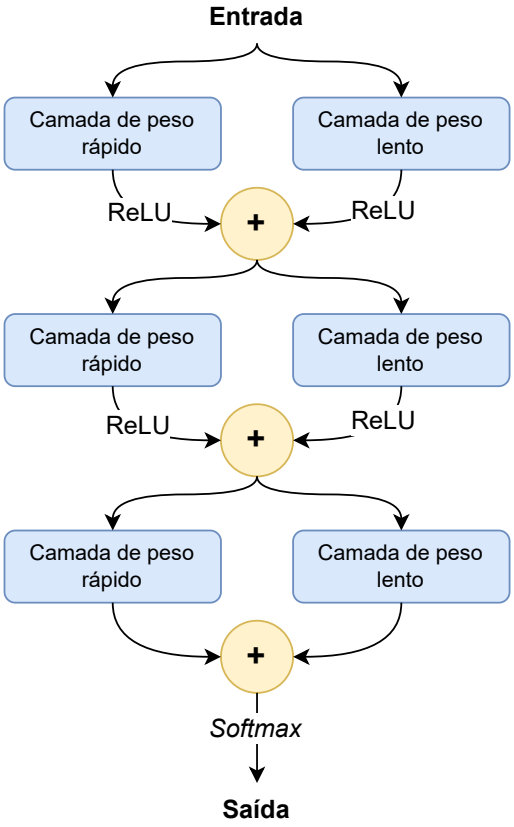


**Figura 3.10. MLP utilizada por [Munkhdalai and Yu 2017] em seus experimentos com *base-learner*.**

de parâmetros, como meta-aprendizado utilizando LSTMs [Ravi and Larochelle 2017], Meta-SGD [Li et al. 2017] e *Reptile* [Nichol and Schulman 2018], esta seção focará no algoritmo MAML[7] [Finn et al. 2017] por sua simplicidade, alto desempenho e facilidade de aplicação.

O algoritmo MAML utiliza procedimentos de otimização simples, como o gradiente descendente estocástico, tanto em sua fase de aprendizado interno (aprendendo tarefas individualmente), quanto em sua fase de aprendizado externo (aprendendo seu meta-modelo). A ideia principal do MAML é aprender bons parâmetros na sua fase de treinamento que ofereçam uma rápida adaptação para novas tarefas. A Figura 3.11 exemplifica como o algoritmo MAML funciona: o parâmetro $\theta$ foi otimizado baseado na otimização individual das tarefas que geraram gradientes $\nabla\mathcal{L}_1$, $\nabla\mathcal{L}_2$ e $\nabla\mathcal{L}_3$. À vista disso, novas tarefas adaptaram facilmente seus parâmetros $\theta_1^*$, $\theta_2^*$ e $\theta_3^*$ a partir de $\theta$.

A Figura 3.12 apresenta um fluxograma para o algoritmo MAML, que recebe como entrada um conjunto de tarefas e a quantidade de passos de treinamento. Inicialmente, é criado um meta-modelo com pesos inicializados aleatoriamente, representado no fluxograma por $\theta$. Em seguida, uma cópia desse modelo $\theta$ para cada uma das tarefas $(\theta_1, \theta_2, \cdots, \theta_N)$ do conjunto de treinamento é criada. Cada uma das tarefas irá treinar seu modelo utilizando como dados de entrada o seu próprio conjunto de suporte e a quanti-

[7] *Model Agnostic Meta-Learning*, ou, em português, Meta-Aprendizado Agnóstico à Modelos.

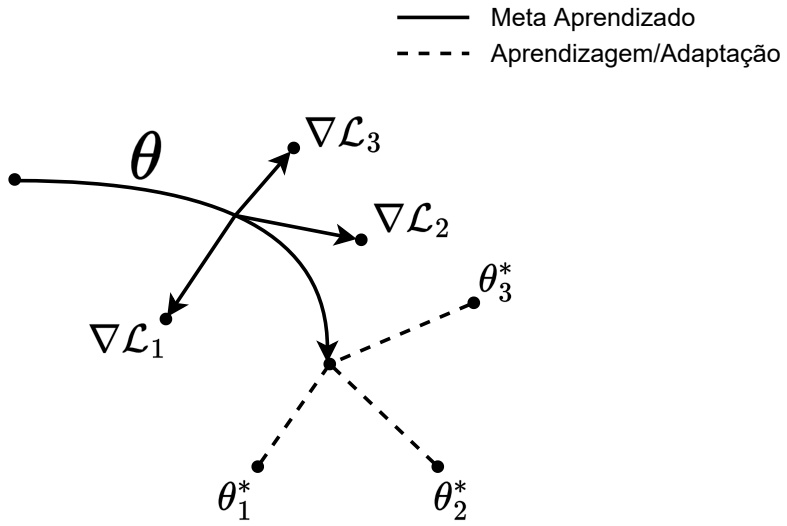


**Figura 3.11. Intuição sobre como o treinamento de meta-modelos baseados em otimização funcionam. Adaptado de [Finn et al. 2017].**

dade de passos de treinamento informada inicialmente, gerando novos pesos atualizados $(\theta_1', \theta_2', \cdots, \theta_N')$.

As tarefas não possuem nenhuma relação entre si em suas fases de treinamento, assemelhando-se a treinamentos de modelos convencionais. Uma vez que os novos pesos foram computados, a função de perda é calculada para cada tarefa utilizando seus respectivos conjuntos de consulta. O meta-modelo utilizará a soma de todas as funções de perda de todas as tarefas treinadas como sua própria função de perda, indicando o quão útil os pesos iniciais no começo do treinamento foram para cada tarefa individualmente. Com a meta-função de perda calculada, os pesos do meta-modelo $\theta$ são atualizados e o fluxograma pode ser repetido quantas vezes forem necessárias.

Dada a sua importância dentre as técnicas de meta-aprendizagem baseadas em otimização, o algoritmo MAML serviu como inspiração para muitos outros trabalhos, como LLAMA [Grant et al. 2018], PLATIPUS [Finn et al. 2018] e BMAML [Yoon et al. 2018]. Entretanto, o MAML possui suas desvantagens: o algoritmo tem custos muito elevados com relação ao tempo de treinamento e consumo de memória, uma vez que precisa computar derivadas de alta-ordem para várias tarefas simultaneamente.

## 3.4. Uso de Meta-Learning em Classificação de Imagens

Esta seção possui uma aplicação prática de todos os conceitos abordados até aqui no capítulo. A Figura 3.13 apresenta um fluxograma do processo de aplicação de uma das técnicas de *meta-learning* abordadas na Seção 3.3: o algoritmo MAML, considerando desde a criação de tarefas do tipo *N-way*, *K-shot* a partir de um conjunto de dados de imagens, até o *fine-tuning* do meta-modelo treinado para uma nova tarefa.

**Descrição do Problema.** Alguns modelos mais recentes para classificação de imagens, como VGG-16 e *ResNet*50, treinados com centenas de milhares de exemplos, conseguem resolver bem diversas tarefas de classificação de imagens mais genéricas, como distinguir

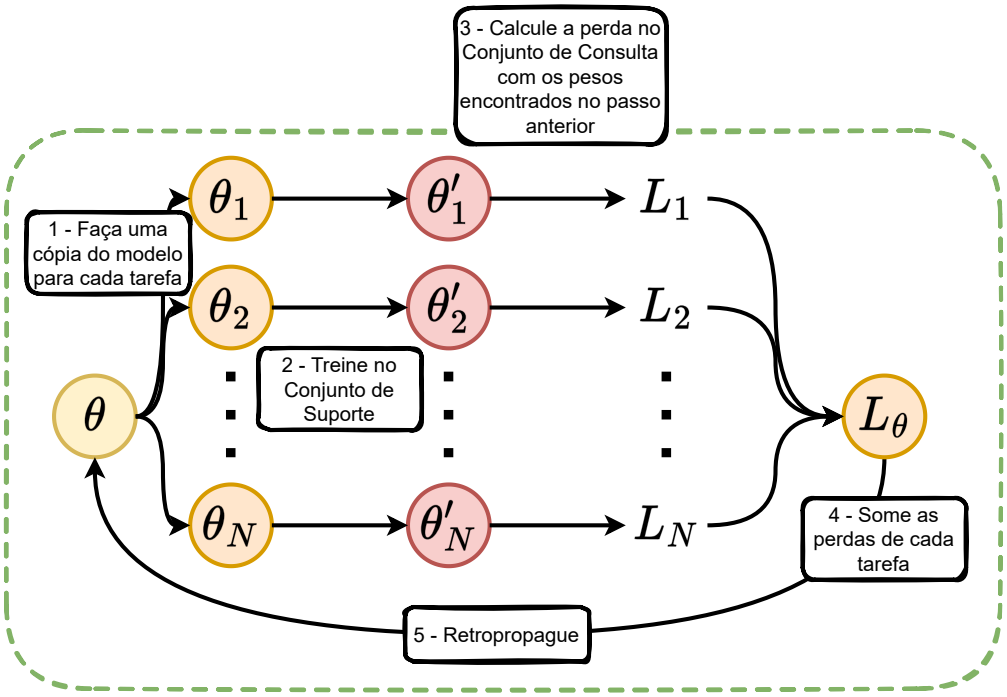


**Figura 3.12. Diagrama que representa o algoritmo MAML.**

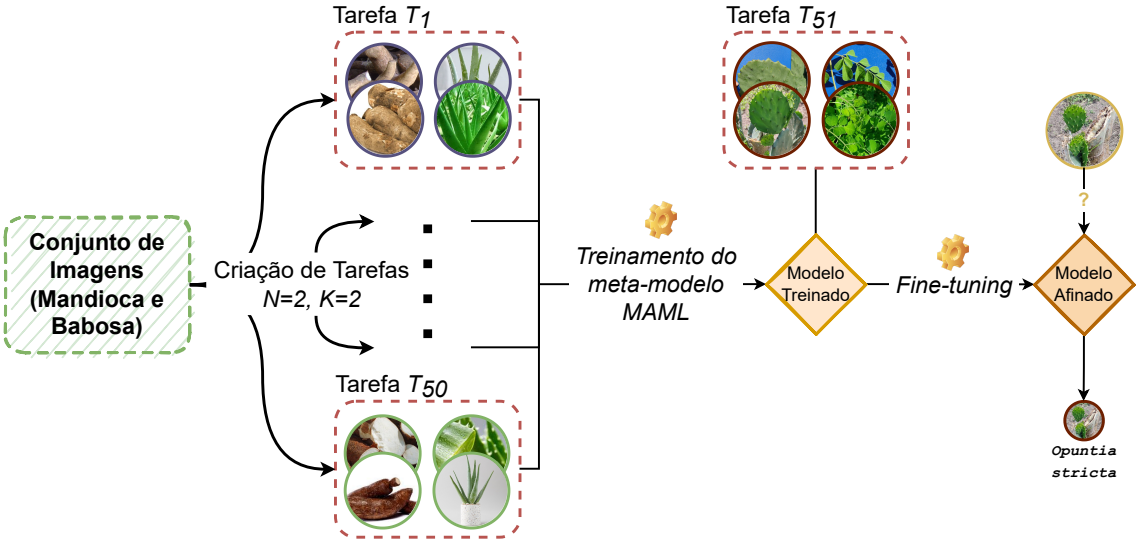


**Figura 3.13. *Pipeline* completo para o treinamento do modelo e *fine-tuning* de um meta-modelo classificador de plantas forrageiras.**

gatos de cachorros, ou leões de cavalos. Entretanto, para domínios específicos, esses modelos podem não fornecer bons resultados.

No contexto da Agricultura de Precisão, classificar imagens de diferentes tipos de plantas forrageiras é uma tarefa importante, pois esses resultados podem auxiliar agricultores na tomada de decisões com relação aos seus cultivos. Infelizmente, não existem opções de conjuntos de dados com quantidades de exemplos relevantes para o treinamento

de, por exemplo, CNNs para classificação de diferentes tipos de forrageiras.

**Conjuntos de Dados Utilizados.** Para resolver essa tarefa, um pequeno conjunto de dados de plantas forrageiras foi criado, contendo seis classes e aproximadamente quinze imagens por classe, e anotado manualmente. Também foi utilizado um subconjunto do conjunto de dados *Plants Type Datasets*[8] contendo 862 imagens do tubérculo mandioca e 1.000 imagens da planta babosa.

Para treinar o meta-modelo, foram criadas cinquenta tarefas do tipo 2-*way*, 2-*shot* a partir do conjunto de dados de mandiocas e babosas. Também foi criada uma tarefa do tipo 2-*way*, 2-*shot* com duas classes diferentes de plantas forrageiras para o processo de *fine-tuning*.

**Treinamento do Meta-Modelo e *Fine-Tuning*.** O Programa 3.7 apresenta o código da criação do meta-modelo que utiliza o algoritmo MAML e as definições de tarefas aplicadas nele. Para a criação do meta-modelo, o modelo base foi uma rede `resnet18` e o otimizador `Adam`. O código apresentado foi escrito utilizando a linguagem *Python* e o *framework Flash*[9]. O Programa 3.8 mostra como instanciar e executar o componente que fará o treinamento do meta-modelo com as tarefas de classificação de mandioca e babosa com o auxílio de uma GPU (caso disponível) e como é feito o processo de *fine-tuning* utilizando a nova tarefa de classificação de forrageiras.

```
model = ImageClassifier(
    backbone="resnet18",

    # Caso queira utilizar a estrategia prototypical networks, use
     ↪  a chave 'prototypicalnetworks' abaixo
    training_strategy="maml",

    pretrained=False,

    # Definição das tarefas que serão utilizadas neste modelo
    training_strategy_kwargs={"epoch_length": 50,
     ↪  "meta_batch_size": 2, "num_tasks": 50, "test_num_tasks":
     ↪  50, "ways": datamodule_fruits.num_classes, "shots": 2,
     ↪  "test_ways": 2, "test_shots": 1},

    optimizer=torch.optim.Adam,
    learning_rate=0.001,
)
```

**Programa 3.7. Criação do meta-modelo que utiliza o algoritmo MAML para tarefas de classificação de plantas (mandioca e babosa).**

**Resultados.** A Tabela 3.1 mostra a comparação de dois modelos de classificação de plantas forrageiras a partir de métricas comuns para este tipo de tarefa. Para o modelo con-

[8] https://www.kaggle.com/datasets/yudhaislamisulistya/plants-type-datasets

[9] https://lightning-flash.readthedocs.io/

```
trainer = flash.Trainer(
    max_epochs=50,
    precision=16,
    accelerator="ddp_shared",
    gpus=int(torch.cuda.is_available()),
    logger=logger,
)

trainer.fit(model, datamodule=datamodule_fruits)

trainer.finetune(model, datamodule=datamodule_forrageiras,
 ↪  strategy="no_freeze")
```

**Programa 3.8. Treinamento do meta-modelo com os dados de mandioca e babosa e *fine-tuning* da tarefa de classificação de forrageiras.**

vencional (*RestNet*-18), a tarefa de classificação de forrageiras foi treinada considerando uma pequena quantidade de dados. O meta-modelo que usa o MAML foi primeiramente treinado com o conjunto de dados de babosas e mandiocas, e um processo de *fine-tuning* foi aplicado para a tarefa de classificação de forrageiras.

| **Modelo** | **Acurácia** | **Precisão** | **Revocação** | **$F_1$-*score*** |
|---|---|---|---|---|
| *ResNet*-18 + MAML | 0.88 | 0.86 | 0.91 | 0.86 |
| *ResNet*-18 | 0.59 | 0.49 | 0.58 | 0.51 |

**Tabela 3.1. Resultados dos dados de teste para o modelo convencional *ResNet*-18 e para o modelo gerado após o *fine-tuning* do meta-modelo criado com o algoritmo MAML e a arquitetura *ResNet* como base (*ResNet*-18 + MAML).**

Conclui-se que, para uma baixa quantidade de dados, o meta-modelo *ResNet*-18 + MAML em comparação com o *ResNet*-18 convencional generaliza bem o aprendizado para tarefas de classificação de babosas e mandiocas, obtendo bons resultados para a tarefa de classificação de forrageiras após o processo de *fine-tuning*.

## 3.5. Limitações, Aplicações e Desafios

A presente seção aborda as limitações dos métodos de *meta-learning*, apresenta algumas aplicações em diferentes áreas e cobre desafios que acompanham esta técnica em ascensão. Além disso, são apresentadas algumas das áreas de pesquisa que merecem serem exploradas por esses métodos.

### 3.5.1. Limitações e Desafios

Muitas aplicações de *meta-learning* apresentam bons resultados quando utilizam uma distribuição de tarefas com alto nível de similaridade como entrada. Quando utilizadas distribuições com tarefas distintas, o conflito de gradientes entre as tarefas pode ocasionar um baixo grau de generalização por parte do meta-modelo [Yu et al. 2020].

Outro desafio que as técnicas de *meta-learning* enfrentam são as potenciais baixas

quantidades de tarefas disponíveis para que o meta-modelo generalize bem para uma nova tarefa. Além disso, os meta-modelos podem não generalizar bem para tarefas que estão fora da distribuição utilizada como treinamento.

Um último desafio a salientar é o custo computacional e de tempo do treinamento dos meta-modelos por conta dos treinamentos aninhados tanto para tarefas quanto para meta-modelo e, a depender da abordagem utilizada, cálculos matemáticos diferenciais envolvidos.

### 3.5.2. Aplicações

Cenários de tarefas com pouca quantidade de dados de treinamento são extremamente desafiadores, principalmente para redes neurais profundas, onde o conjunto de dados é fator determinante no desempenho de um modelo. Em geral, quando tais modelos são treinados com conjuntos de dados pequenos, obtém-se um *overfitting* do modelo ou sua não-convergência. Abordagens baseadas em *meta-learning* são úteis em diversas áreas, pois todas podem sofrer com os mesmos problemas de disponibilidade de dados.

**Visão Computacional.** As aplicações mais comuns nessa área envolvem tarefas como reconhecimento de imagens [Snell et al. 2017], detecção de objetos [Kang et al. 2019], segmentação de objetos [Dong and Xing 2018], e geração de imagens [Zakharov et al. 2019] e vídeos [Wang et al. 2019]. Infelizmente, os resultados obtidos com técnicas de *meta-learning* aplicadas à Visão Computacional ainda não se equiparam ao treinamento de modelos de Aprendizado Profundo com uma grande quantidade de dados, mas mostram-se úteis por utilizarem conjuntos de dados pequenos, como um conjunto de poucas imagens radiológicas para detecção de doenças pulmonares.

**Processamento de Linguagem Natural (PLN).** Pesquisas vêm sendo desenvolvidas em tarefas relacionadas a modelagem de linguagem natural, como classificação de textos [Bao et al. 2019], tradução automática de textos [Gu et al. 2018] e reconhecimento de fala [Hsu et al. 2020]. Um dos grandes desafios enfrentados no treinamento de tarefas de PLN é o baixo volume de dados rotulados para treinamento em domínios específicos, como textos relacionados à segurança pública ou notícias, e línguas extintas em tarefas de tradução de textos.

**Bem-Estar Social.** Técnicas de *meta-learning* podem ser utilizadas em tarefas que melhoram o estado de bem-estar social e que possuem uma quantidade muito escassa de dados, como classificação de imagens médicas [Maicas et al. 2018, Mirikharaji et al. 2019], descoberta de medicamentos [Altae-Tran et al. 2017], predições do comportamento de pandemias [Panagopoulos et al. 2021] e identificação de *fake news* [Salem et al. 2021].

### 3.5.3. Diferença Entre *Meta-Learning* e Outras Áreas

Além de *meta-learning*, outras técnicas, como *transfer learning* e *multi-task learning*, também aproveitam da ideia de utilizar conhecimentos prévios para o treinamento de novas tarefas de forma mais rápida ao invés de treinar uma nova tarefa do zero. Como os limites entre essas áreas nem sempre são claros, esta subseção apresenta uma explicação breve de cada uma e mostra uma comparação entre elas, ressaltando suas vantagens e desvantagens.

***Transfer Learning.*** Nessa abordagem, explora-se o que já foi aprendido em uma determinada tarefa base, que pode já possuir um bom modelo treinado em um volume de dados relevante, para melhorar o aprendizado de uma nova tarefa que, por exemplo, não possui dados suficientes para treinar um bom modelo. A transferência do aprendizado pode se dar de várias formas, como utilizando um *feature extraction*, que reutiliza camadas e parâmetros do modelo treinado na tarefa base, ou um *fine-tuning* dos parâmetros de um modelo pré-treinado para uma nova tarefa. Considere a seguinte situação como exemplo de *transfer learning*: suponha que uma tarefa base de classificação de espécies de passarinhos que possui um grande volume de dados rotulados. Uma tarefa alvo, com pouca quantidade de dados rotulados, pode ser outra tarefa de classificação de pinguins e avestruzes. O conhecimento aprendido no treinamento da tarefa base pode ser transferido, reutilizando algumas camadas do modelo treinado e adicionando uma nova camada de classificação e, se feito corretamente, pode melhorar o desempenho da tarefa alvo[10].

***Multi-Task Learning.*** Nessa abordagem, um conjunto de tarefas base são utilizadas no treinamento de um modelo único, baseado em uma arquitetura personalizada, que consiga generalizar bem o suficiente para várias tarefas em simultâneo. Um exemplo de *multi-task learning* seria o treinamento de um modelo utilizando como dados de entrada imagens de paisagens da natureza. Esse modelo deve conseguir generalizar bem para diversas tarefas envolvendo esse conjunto de dados, como segmentação semântica de pixeis, classificação da imagem, estimativa de profundidade, etc. Diferente do *transfer learning*, a abordagem do *multi-task learning* aprende várias tarefas simultaneamente, enquanto a primeira treina uma tarefa base inicialmente e, em seguida, transfere o aprendizado para uma tarefa alvo.

***Meta-Learning.*** Também conhecido como um conjunto de algoritmos que cria modelos que "aprendem a aprender", esse paradigma visa o aprendizado de uma determinada tarefa alvo, explorando o aprendizado adquirido a partir de um conjunto de tarefas-base treinadas individualmente. Um exemplo do uso de *meta-learning* seria o treinamento de uma tarefa de classificação de gatos e cachorros utilizando o meta-conhecimento adquirido a partir do treinamento de outras tarefas-base, como uma tarefa de classificação de coelhos e lebres e uma classificação de leões e tigres. Quando *meta-learning* e *transfer learning* são comparadas, a maior diferença surge no procedimento de otimização dos modelos: nas tarefas que envolvem *transfer learning*, não existe um meta-objetivo a ser otimizado, ao contrário do *meta-learning*, que otimiza um meta-objetivo a partir da otimização das tarefas-base utilizadas na fase de treinamento. Já *meta-learning* comparado com *multi-task learning*, a mesma diferença com relação à otimização de modelos citada é anteriormente repetida; além disso, *multi-task learning* consegue lidar com tarefas heterogêneas, ao contrário do *meta-learning*, que lida apenas com tarefas homogêneas (como, por exemplo, apenas tarefas de classificação binária ou tarefas de regressão).

O Quadro 3.1 apresenta um resumo dos três paradigmas apresentados, enfatizando suas vantagens e desvantagens principais.

[10] *Transfer learning* parece um pouco com o processo de *fine-tuning*, mas possuem duas perspectivas diferentes: *transfer learning* ocorre quando um modelo desenvolvido para uma tarefa é reutilizado para outra tarefa, enquanto *fine-tuning* é uma abordagem que treina novamente um modelo treinado para se adequar a uma nova tarefa.

| Paradigma | Vantagens | Desvantagens |
|---|---|---|
| ***Transfer Learning*** | - Precisa de uma menor quantidade de dados para a tarefa alvo,<br>- Bom para extração de features. | - Modelos pré-treinados tendem a se ajustar demais à tarefa alvo,<br>- Em geral utiliza modelos base complexos, com milhões de parâmetros, nem sempre necessários para tarefas alvo,<br>- Concentra-se apenas em melhorar o desempenho da tarefa alvo. |
| ***Multi-task Learning*** | - Consegue lidar com tarefas heterogêneas e entradas multimodais,<br>- Visa melhorar o desempenho de todas as tarefas ao mesmo tempo. | - Exige um grande conjunto de dados para treinamento,<br>- Ao adicionar uma nova tarefa, é necessário treinar novamente a rede, pois pode exigir alterações de arquitetura. |
| ***Meta Learning*** | - Fácil adição de novas tarefas,<br>- Menor quantidade de dados de treinamento necessária para uma nova tarefa.<br>- Generalização robusta entre tarefas,<br>- Utiliza um objetivo de meta learning (otimização de dois níveis). | - Funciona apenas para tarefas homogêneas, pois as tarefas de base e alvo devem corresponder,<br>- Concentra-se apenas em melhorar o desempenho da tarefa alvo. |

**Quadro 3.1. Comparativo entre *Transfer Learning*, *Multi-task Learning* e *Meta-Learning*. Adaptado de [Upadhyay et al. 2021].**

## 3.6. Conclusão

O presente capítulo apresentou como técnicas de *meta-learning* podem ser utilizadas no contexto de algoritmos de Aprendizado Profundo. Inicialmente, foram apresentados alguns conceitos básicos sobre Aprendizado Profundo e algumas arquiteturas comuns para tarefas de classificação de imagens[11], como as Redes Neurais Convolucionais (CNNs). Em seguida, o conceito de *meta-learning* foi definido, com suas diferentes fases, e a principal forma de organização de tarefas a serem utilizadas pelos métodos (*N-way*, *K-shot*) foi apresentada. Uma comparação entre outros tipos de aprendizado também foi introduzida (*Transfer Learning*, *Multi-task Learning* e *Meta-Learning*), enfatizando os aspectos positivos e negativos de cada uma delas. Por fim, alguns dos diversos métodos de *meta-learning* foram apresentados, divididos em métodos baseados em métricas, modelos e otimização.

[11]Tarefa escolhida para exemplificar como *meta-learning* funciona ao longo do trabalho.

## Referências

[Altae-Tran et al. 2017] Altae-Tran, H., Ramsundar, B., Pappu, A. S., and Pande, V. (2017). Low data drug discovery with one-shot learning. *ACS central science*, 3(4):283–293.

[Bao et al. 2019] Bao, Y., Wu, M., Chang, S., and Barzilay, R. (2019). Few-shot text classification with distributional signatures. *arXiv preprint arXiv:1908.06039*.

[Chollet 2021] Chollet, F. (2021). *Deep learning with Python*. Simon and Schuster.

[Dong and Xing 2018] Dong, N. and Xing, E. P. (2018). Few-Shot Semantic Segmentation with Prototype Learning. In *BMVC*, volume 3.

[Dumoulin and Visin 2016] Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

[Finn et al. 2017] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

[Finn et al. 2018] Finn, C., Xu, K., and Levine, S. (2018). Probabilistic model-agnostic meta-learning. *Advances in neural information processing systems*, 31.

[Garcia and Bruna 2017] Garcia, V. and Bruna, J. (2017). Few-shot learning with graph neural networks. *arXiv preprint arXiv:1711.04043*.

[Garnelo et al. 2018] Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D., and Eslami, S. A. (2018). Conditional neural processes. In *International Conference on Machine Learning*, pages 1704–1713. PMLR.

[Grant et al. 2018] Grant, E., Finn, C., Levine, S., Darrell, T., and Griffiths, T. (2018). Recasting gradient-based meta-learning as hierarchical bayes. *arXiv preprint arXiv:1801.08930*.

[Gu et al. 2018] Gu, J., Wang, Y., Chen, Y., Cho, K., and Li, V. O. (2018). Meta-learning for low-resource neural machine translation. *arXiv preprint arXiv:1808.08437*.

[He et al. 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hochreiter and Schmidhuber 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Hsu et al. 2020] Hsu, J.-Y., Chen, Y.-J., and Lee, H.-y. (2020). Meta learning for end-to-end low-resource speech recognition. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7844–7848. IEEE.

[Huisman et al. 2021] Huisman, M., Van Rijn, J. N., and Plaat, A. (2021). A survey of deep meta-learning. *Artificial Intelligence Review*, 54(6):4483–4541.

[Kang et al. 2019] Kang, B., Liu, Z., Wang, X., Yu, F., Feng, J., and Darrell, T. (2019). Few-shot object detection via feature reweighting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8420–8429.

[Koch et al. 2015] Koch, G., Zemel, R., Salakhutdinov, R., et al. (2015). Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2, page 0. Lille.

[Lake et al. 2017] Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and brain sciences*, 40.

[Li et al. 2017] Li, Z., Zhou, F., Chen, F., and Li, H. (2017). Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*.

[Maicas et al. 2018] Maicas, G., Bradley, A. P., Nascimento, J. C., Reid, I., and Carneiro, G. (2018). Training medical image analysis systems like radiologists. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 546–554. Springer.

[Mirikharaji et al. 2019] Mirikharaji, Z., Yan, Y., and Hamarneh, G. (2019). Learning to segment skin lesions from noisy annotations. In *Domain adaptation and representation transfer and medical image learning with less labels and imperfect data*, pages 207–215. Springer.

[Mishra et al. 2017] Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2017). A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*.

[Munkhdalai and Yu 2017] Munkhdalai, T. and Yu, H. (2017). Meta networks. In *International Conference on Machine Learning*, pages 2554–2563. PMLR.

[Neves Oliveira et al. 2022a] Neves Oliveira, B. S., do Rêgo, L. G. C., Peres, L., da Silva, T. L. C., and de Macêdo, J. A. F. (2022a). Processamento de linguagem natural via aprendizagem profunda. *Sociedade Brasileira de Computação*.

[Neves Oliveira et al. 2022b] Neves Oliveira, B. S., do Rêgo, L. G. C., Peres, L., da Silva, T. L. C., and de Macêdo, J. A. F. (2022b). Processamento de Linguagem Natural via Aprendizagem Profunda. *Sociedade Brasileira de Computação*.

[Nichol and Schulman 2018] Nichol, A. and Schulman, J. (2018). Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2(3):4.

[Panagopoulos et al. 2021] Panagopoulos, G., Nikolentzos, G., and Vazirgiannis, M. (2021). Transfer graph neural networks for pandemic forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4838–4845.

[Ravi and Larochelle 2017] Ravi, S. and Larochelle, H. (2017). Optimization as a model for few-shot learning. In *International Conference on Learning Representations*.

[Salem et al. 2021] Salem, F. K. A., Al Feel, R., Elbassuoni, S., Ghannam, H., Jaber, M., and Farah, M. (2021). Meta-learning for fake news detection surrounding the syrian war. *Patterns*, 2(11):100369.

[Schmidhuber 1987] Schmidhuber, J. (1987). *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München.

[Snell et al. 2017] Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30.

[Thrun 1998] Thrun, S. (1998). Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer.

[Upadhyay et al. 2021] Upadhyay, R., Phlypo, R., Saini, R., and Liwicki, M. (2021). Sharing to learn and learning to share-fitting together meta-learning, multi-task learning, and transfer learning: A meta review. *arXiv preprint arXiv:2111.12146*.

[Vinyals et al. 2016] Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. (2016). Matching networks for one shot learning. *Advances in neural information processing systems*, 29.

[Visca et al. 2022] Visca, M., Powell, R., Gao, Y., and Fallah, S. (2022). Deep meta-learning energy-aware path planner for unmanned ground vehicles in unknown terrains. *IEEE Access*, 10:30055–30068.

[Wang et al. 2022] Wang, C., Zhu, Y., Liu, H., Zang, T., Yu, J., and Tang, F. (2022). Deep meta-learning in recommendation systems: A survey. *arXiv preprint arXiv:2206.04415*.

[Wang et al. 2019] Wang, T.-C., Liu, M.-Y., Tao, A., Liu, G., Kautz, J., and Catanzaro, B. (2019). Few-shot video-to-video synthesis. *arXiv preprint arXiv:1910.12713*.

[Yoon et al. 2018] Yoon, J., Kim, T., Dia, O., Kim, S., Bengio, Y., and Ahn, S. (2018). Bayesian model-agnostic meta-learning. *Advances in neural information processing systems*, 31.

[Yu et al. 2020] Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. (2020). Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836.

[Zakharov et al. 2019] Zakharov, E., Shysheya, A., Burkov, E., and Lempitsky, V. (2019). Few-shot adversarial learning of realistic neural talking head models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9459–9468.

[Zhang et al. 2021] Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. (2021). Dive into Deep Learning. *arXiv preprint arXiv:2106.11342*.