Relatório Projeto – Meta II

Computação SWARM

Inteligência Computacional

Lucas Pagnano Golobovante - 2023140728
Diogo Valdemar Melo Estimado - 2023130197

Índice

Introdução 2
Computação SWARM 2
Aplicação no treino de redes neuronais 2
Algoritmo Gray Wolf Otimization (GWO) 3
Comparação com o Particle Swarm Optimization (PSO) 3
Vantagens e Desvantagens 4
Benchmark – Função Ackley 4
Implementação 4
Configuração 5
Resultados 5
D=2 5
D=3 5
Otimização de híper-parâmetros 6
Configuração 6
Resultados da otimização 7
Treino e validação do modelo final 8
Híper-parâmetros PSO 8
Híper-parâmetros GWO 9
Conclusão 11

Lucas Pagnano Golobovante – 2023140728
Diogo Valdemar Melo Estimado - 2023130197

Introdução

O projeto geral tem como objetivo desenvolver um sistema de Deep Learning capaz de categorizar automaticamente resíduos em cinco classes: Metal, Orgânico, Papel, Plástico e Vidro.

Na Fase I do projeto, foi desenvolvida uma Rede Neuronal Convolucional (CNN) base. Este modelo inicial atingiu uma accuracy de 70% no conjunto de teste. No entanto, a análise do treino revelou um overfitting significativo, com a accuracy de treino (82%) e a accuracy de validação (66%), indicando que a arquitetura do modelo não estava otimizada para a generalização.

O objetivo desta Fase II é resolver esta limitação através da otimização de híper-parâmetros. Em vez de recorrer a métodos manuais, de grelha (exaustivos) ou aleatórios (sem garantia de qualidade), esta meta implementa um algoritmo de Swarm Intelligence para encontrar uma configuração de rede superior.

O algoritmo selecionado foi o Gray Wolf Optimization (GWO), que será usado para otimizar três híper-parâmetros: a taxa de aprendizagem (learning rate), a taxa de dropout e o número de neurónios na camada densa. O desempenho do GWO será ainda comparado com o algoritmo de Swarm base, o Particle Swarm Optimization (PSO), para validar a eficácia da otimização.

Computação SWARM

A Computação Swarm é um paradigma de computação inspirado na natureza, focado em estudar o comportamento coletivo de sistemas descentralizados e auto-organizados. A ideia central é que, embora os agentes individuais (como uma formiga, uma abelha ou um lobo) possam seguir regras muito simples e ter conhecimento local limitado, a interação entre os milhares de agentes do "enxame" (swarm) permite que o sistema como um todo exiba um comportamento global inteligente e resolva problemas complexos.

Em vez de uma entidade centralizada que coordena todas as ações, a inteligência "emerge" das interações locais entre os agentes e entre os agentes e o seu ambiente.

Aplicação no treino de redes neuronais

No contexto do treino de uma rede neuronal, a computação SWARM é extremamente eficaz para o problema da Otimização de Híper-parâmetros.

Os algoritmos de Swarm tratam cada conjunto de híper-parâmetros como um "agente" (uma partícula ou lobo) a "voar" pelo espaço de procura. Estes agentes partilham informação sobre as áreas que encontraram com bons resultados (baixo custo), permitindo ao enxame convergir para soluções ótimas de forma muito mais eficiente do que uma pesquisa exaustiva ou aleatória.

Lucas Pagnano Golobovante – 2023140728
Diogo Valdemar Melo Estimado - 2023130197

Algoritmo Gray Wolf Otimization (GWO)

O GWO é um algoritmo de otimização meta-heurístico inspirado na estrutura social e no comportamento de caça das alcateias de lobos cinzentos.

O GWO simula a hierarquia social de uma alcateia para guiar a procura pela solução ótima. Esta hierarquia é dividida em quatro níveis:

1. Alfa (α): O líder da alcateia. No contexto da otimização, esta é a melhor solução (o melhor conjunto de híper-parâmetros) encontrada até ao momento.
2. Beta (β): O segundo no comando, que auxilia o Alfa. Esta é a segunda melhor solução encontrada.
3. Delta (δ): O terceiro melhor, submisso ao Alfa e Beta. Esta é a terceira melhor solução encontrada.
4. Ómega (ω): Os restantes lobos da alcateia. No algoritmo, estes são todos os outros agentes de procura (soluções candidatas) que seguem as posições dos três líderes.

O processo de otimização simula a "caça", que é dividida em três fases:

1. Procurar a presa: A alcateia espalha-se para explorar o espaço de procura.
2. Cercar a presa: Os lobos Ómega atualizam as suas posições, aproximando-se das posições estimadas da "presa".
3. Atacar a presa: A "presa" é a solução ótima. O GWO assume que os lobos Alfa, Beta e Delta (as 3 melhores soluções atuais) têm a melhor informação sobre a localização da presa.

Assim, em cada iteração, todos os lobos Ómega atualizam a sua posição no espaço de procura calculando uma nova posição que é uma média das posições dos seus três líderes (Alfa, Beta e Delta), com um fator de aleatoriedade para simular a exploração de novas áreas.

Comparação com o Particle Swarm Optimization (PSO)

O PSO é inspirado em bandos de pássaros à procura de comida. Cada "partícula" (solução candidata) ajusta o seu "voo" (trajetória no espaço de procura) com base em duas informações principais:

1. A sua melhor posição pessoal (pbest): A "memória" da partícula do melhor local que ela própria já encontrou.
2. A melhor posição global (gbest): O melhor local encontrado por qualquer partícula em todo o enxame.

A nova posição da partícula é um vetor influenciado tanto pela sua própria experiência como pela experiência social do enxame.

Lucas Pagnano Golobovante – 2023140728
Diogo Valdemar Melo Estimado - 2023130197

Vantagens e Desvantagens

A principal diferença entre os dois é como a informação social é partilhada:

- O PSO baseia-se em duas fontes de informação (pbest e gbest).
- O GWO baseia-se em três fontes de informação (Alfa, Beta e Delta) e não tem a "memória" pessoal do pbest.

O GWO ao ser guiado por três das melhores soluções (em vez de apenas uma, como o gbest do PSO), tem um comportamento de exploração superior. É menos provável que o enxame inteiro convirja prematuramente para um ótimo local.

Entretanto, a ausência de memória (pbest) pode fazer com que o GWO "esqueça" boas áreas que explorou. Em problemas mais simples, o PSO pode por vezes convergir mais rapidamente para a solução final, pois as suas partículas são fortemente atraídas pelo gbest.

Como no nosso projeto o problema é mais complexo e com muitos ótimos locais, a capacidade de exploração superior do GWO é uma vantagem teórica significativa.

Benchmark – Função Ackley

Antes de aplicar os algoritmos de otimização à rede neuronal, foi realizado um teste de benchmark numa função matemática padrão. A função escolhida foi a Função de Ackley, uma função complexa com muitos mínimos locais, mas com um único mínimo global conhecido no ponto f(0, 0, ..., 0) = 0.

O objetivo deste teste foi validar a implementação dos algoritmos GWO e PSO e comparar a sua capacidade de escapar de mínimos locais para encontrar o ótimo global.

Implementação

A função de Ackley foi implementada em Python, utilizando a biblioteca NumPy para os cálculos vetoriais. Esta função serviu como a função de fitness que os algoritmos tentaram minimizar.

```python
def ackley(solution):
    n = len(solution)
    sum1 = np.sum(solution**2)
    sum2 = np.sum(np.cos(2 * np.pi * solution))

    term1 = -20 * np.exp(-0.2 * np.sqrt(sum1 / n))
    term2 = -np.exp(sum2 / n)

    return term1 + term2 + 20 + np.exp(1)
```

Configuração

Ambos os algoritmos, GWO e PSO, foram configurados para procurar o mínimo da função com uma população de 30 agentes e um total de 100 iterações. Os testes foram realizados para as dimensões D=2 e D=3, com os limites de procura definidos entre [-32, 32].

```
D = 3
n_agents = 30
n_iterations = 100

lb = [-32] * D
ub = [32] * D
```

```
D = 2
n_agents = 30
n_iterations = 100

lb = [-32] * D
ub = [32] * D
```

```
pso_optimizer = pso(n_agents, ackley, lb, ub, D, n_iterations)
```

```
gwo_optimizer = gwo(n_agents, ackley, lb, ub, D, n_iterations)
```

Resultados

D=2

```
Agentes: 30 | Iterações: 100

A executar PSO...
PSO - Melhor Fitness (Valor da função): 0.00
PSO - Melhor Solução (Posição): [-0. -0.]

A executar GWO...
GWO - Melhor Fitness (Valor da função): 0.00
GWO - Melhor Solução (Posição): [ 0. -0.]
```

D=3

```
Agentes: 30 | Iterações: 100

A executar PSO...
PSO - Melhor Fitness (Valor da função): 0.00
PSO - Melhor Solução (Posição): [-0.  0. -0.]

A executar GWO...
GWO - Melhor Fitness (Valor da função): 0.00
GWO - Melhor Solução (Posição): [0. 0. 0.]
```

Com isso podemos observar que tanto o GWO como o PSO foram capazes de convergir para o ótimo global (Fitness = 0.00) em ambas as dimensões. Os resultados validam que ambos os algoritmos estão corretamente implementados e são plenamente capazes de navegar num espaço de procura complexo para encontrar a solução ideal.

Lucas Pagnano Golobovante – 2023140728
Diogo Valdemar Melo Estimado - 2023130197

Otimização de híper-parâmetros

Depois da validação dos algoritmos no benchmark, o GWO e o PSO foram aplicados a otimização da arquitetura da Rede Neuronal Convolucional (CNN) realizada na Fase I.

Configuração

O objetivo foi encontrar a combinação de híper-parâmetros que maximizasse a validation accuracy do modelo. Para isso, escolhemos três híper-parâmetros:

1. Learning Rate: Controla a magnitude do ajuste dos pesos da rede durante o treino.
2. Dropout Rate: A percentagem de neurónios desligados aleatoriamente na camada densa para combater o overfitting.
3. Dense Neurons: A capacidade da principal camada de classificação do modelo.

Os limites de valores para cada híper-parâmetro foram:

```
D = 3
lb = [0.0001, 0.2, 64]
ub = [0.01,   0.6, 256]
```

Para que os algoritmos de Swarm pudessem avaliar a qualidade de cada solução, foi criada uma função de fitness. Esta função recebe uma solução do algoritmo (ex: [0.001, 0.3, 128]), constrói o modelo Keras correspondente e treina-o durante 5 épocas (`EPOCHS_POR_TESTE = 5`).

Além disso, utilizamos 8 agentes e 8 iterações.

```
N_AGENTS = 8
N_ITERATIONS = 8
```

Como os algoritmos de Swarm (GWO e PSO) são desenhados para minimizar um valor, a função retorna um custo, que foi definido como (1.0 - validation_accuracy). O algoritmo que encontrar o menor custo, encontrou a melhor accuracy.

Lucas Pagnano Golobovante – 2023140728
Diogo Valdemar Melo Estimado - 2023130197

```python
def cnn_fitness_function(solution):

    learning_rate = solution[0]
    dropout_rate  = solution[1]
    dense_neurons = int(solution[2])

    print(f"\n--- A Testar: LR={learning_rate:.6f} | Dropout={dropout_rate:.2f} | Neurons={dense_neurons} ---")

    try:
        tf.keras.backend.clear_session()

        model = models.Sequential([
            layers.Conv2D(32, (3,3), activation="relu", input_shape=(128,128,3)),
            layers.MaxPooling2D((2,2)),
            layers.Conv2D(64, (3,3), activation="relu"),
            layers.MaxPooling2D((2,2)),
            layers.Conv2D(128, (3,3), activation="relu"),
            layers.MaxPooling2D((2,2)),
            layers.Flatten(),
            layers.Dense(dense_neurons, activation="relu"),
            layers.Dropout(dropout_rate),
            layers.Dense(5, activation="softmax")
        ])

        optimizer = Adam(learning_rate=learning_rate)
        model.compile(optimizer=optimizer,
                      loss="categorical_crossentropy",
                      metrics=["accuracy"])

        history = model.fit(
            train_gen,
            epochs=EPOCHS_POR_TESTE,
            validation_data=val_gen,
            verbose=0
        )

        val_accuracy = history.history['val_accuracy'][-1]
        fitness_score = 1.0 - val_accuracy

        print(f"--- Resultado: Val_Acc: {val_accuracy:.4f} | Custo (Fitness): {fitness_score:.4f} ---")

    except Exception as e:
        print(f"!!! Erro ao treinar: {e}. A descartar solução. !!!")
        fitness_score = 10.0

    return fitness_score
```

Resultados da otimização

```
--- OTIMIZAÇÃO PSO CONCLUÍDA (Tempo: 287.81 minutos) ---
Melhor Custo (1 - Val_Acc): 0.3672
Melhores Hiperparâmetros (PSO):
  Learning Rate: 0.000954
  Dropout Rate:  0.52
  Dense Neurons: 162
```

```
--- OTIMIZAÇÃO GWO CONCLUÍDA (Tempo: 308.80 minutos) ---
Melhor Custo (1 - Val_Acc): 0.3422
Melhores Hiperparâmetros (GWO):
  Learning Rate: 0.000625
  Dropout Rate:  0.37
  Dense Neurons: 140
```

A análise desta fase mostra que o GWO foi mais eficaz, encontrando um conjunto de híper-parâmetros que resultou num custo menor (accuracy de validação superior) do que o PSO.

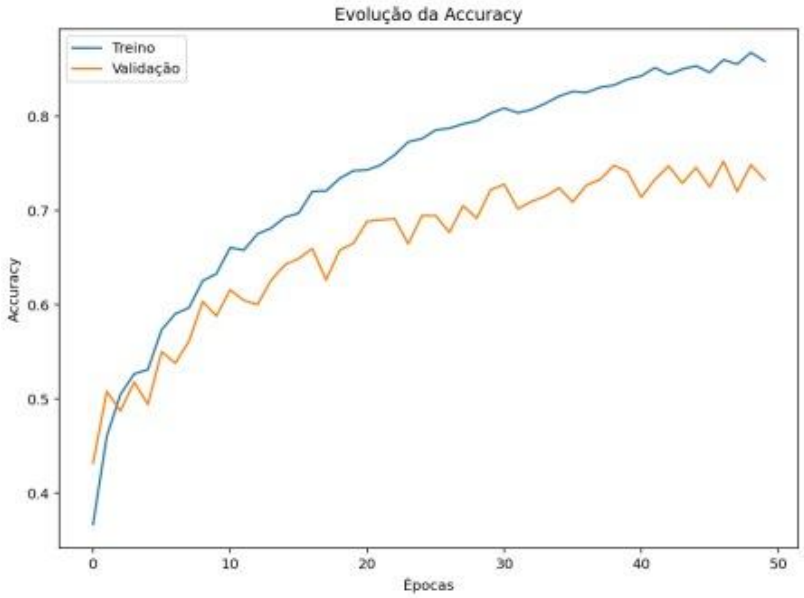
Treino e validação do modelo final

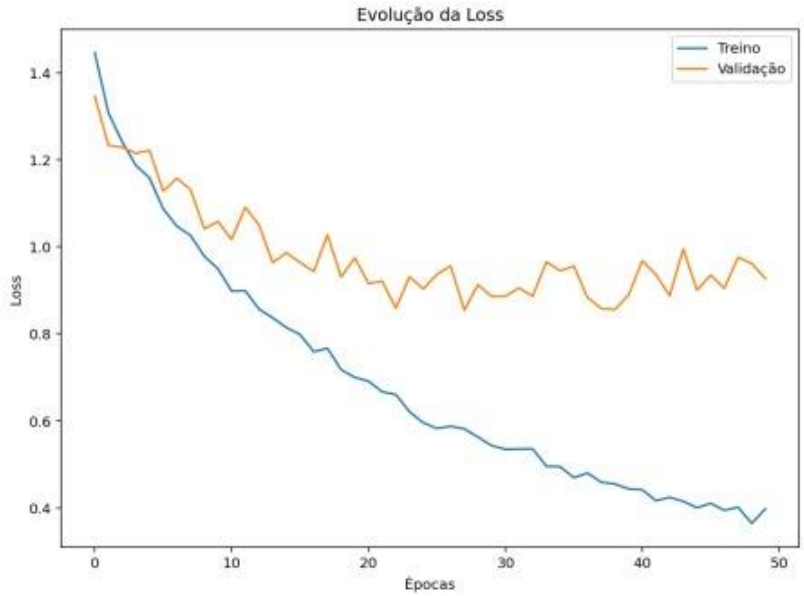
Os dois conjuntos de híper-parâmetros encontrados foram então usados para treinar dois modelos finais, desta vez durante 50 épocas completas, para avaliar o seu verdadeiro desempenho.

Híper-parâmetros PSO

Treino: (LR=0.000954, Dropout=0.52, Neurons=162)

```
Epoch 45/50
225/225 ━━━━━━━━━━━━━━━━━━━━ 98s 434ms/step - accuracy: 0.8533 - loss: 0.4001 - val_accuracy: 0.7456 - val_loss: 0.8998
Epoch 46/50
225/225 ━━━━━━━━━━━━━━━━━━━━ 100s 443ms/step - accuracy: 0.8465 - loss: 0.4102 - val_accuracy: 0.7250 - val_loss: 0.9344
Epoch 47/50
225/225 ━━━━━━━━━━━━━━━━━━━━ 98s 437ms/step - accuracy: 0.8600 - loss: 0.3940 - val_accuracy: 0.7522 - val_loss: 0.9042
Epoch 48/50
225/225 ━━━━━━━━━━━━━━━━━━━━ 101s 450ms/step - accuracy: 0.8553 - loss: 0.4015 - val_accuracy: 0.7200 - val_loss: 0.9752
Epoch 49/50
225/225 ━━━━━━━━━━━━━━━━━━━━ 93s 410ms/step - accuracy: 0.8676 - loss: 0.3643 - val_accuracy: 0.7483 - val_loss: 0.9610
Epoch 50/50
225/225 ━━━━━━━━━━━━━━━━━━━━ 88s 392ms/step - accuracy: 0.8585 - loss: 0.3975 - val_accuracy: 0.7328 - val_loss: 0.9269
```

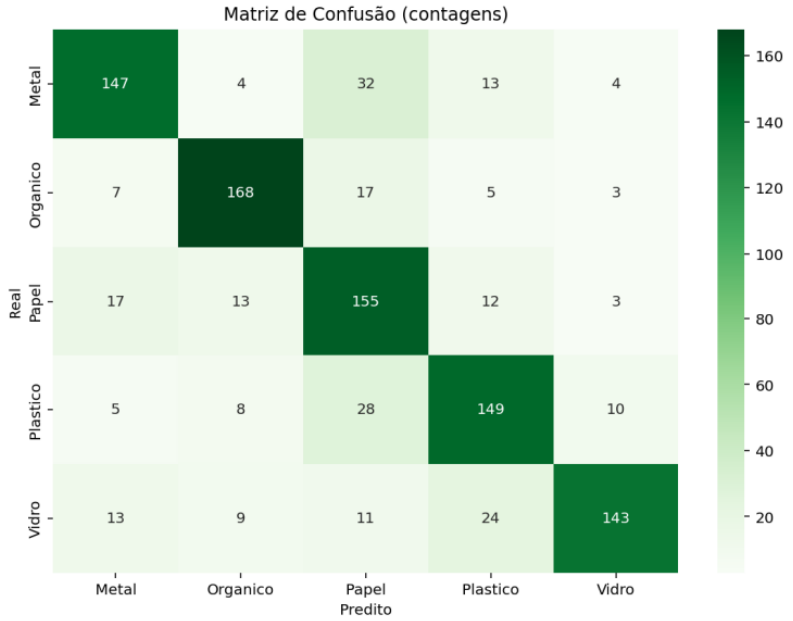




Teste:

	precision	recall	f1-score	support
Metal	0.78	0.73	0.76	200
Organico	0.83	0.84	0.84	200
Papel	0.64	0.78	0.70	200
Plastico	0.73	0.74	0.74	200
Vidro	0.88	0.71	0.79	200
accuracy			0.76	1000
macro avg	0.77	0.76	0.76	1000
weighted avg	0.77	0.76	0.76	1000

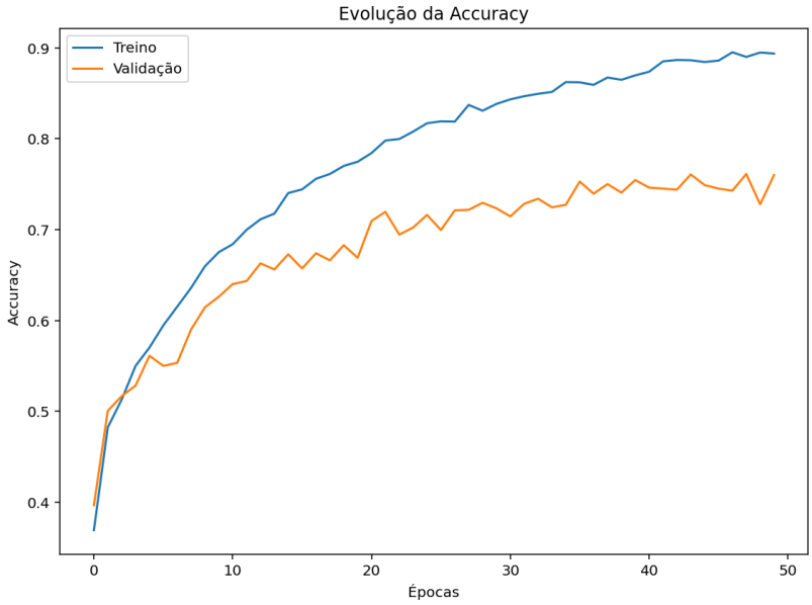
Especificidade média (macro): 0.9405
AUC média (macro): 0.9395

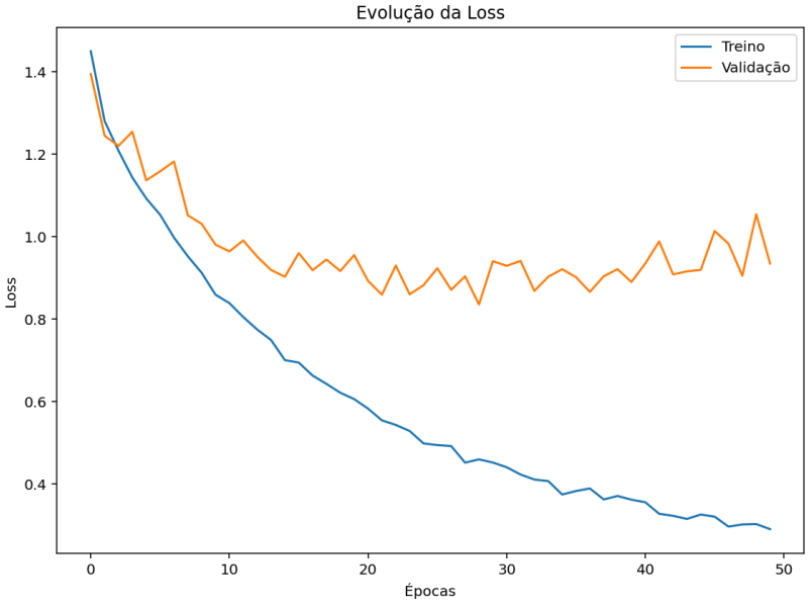


Híper-parâmetros GWO

Treino: (LR=0.000625, Dropout=0.37, Neurons=140)

```
Epoch 45/50
225/225 ━━━━━━━━━━━━━━━━━━━━ 58s 257ms/step - accuracy: 0.8843 - loss: 0.3248 - val_accuracy: 0.7489 - val_loss: 0.9187
Epoch 46/50
225/225 ━━━━━━━━━━━━━━━━━━━━ 57s 252ms/step - accuracy: 0.8858 - loss: 0.3196 - val_accuracy: 0.7450 - val_loss: 1.0129
Epoch 47/50
225/225 ━━━━━━━━━━━━━━━━━━━━ 58s 256ms/step - accuracy: 0.8950 - loss: 0.2956 - val_accuracy: 0.7428 - val_loss: 0.9823
Epoch 48/50
225/225 ━━━━━━━━━━━━━━━━━━━━ 57s 255ms/step - accuracy: 0.8899 - loss: 0.3007 - val_accuracy: 0.7611 - val_loss: 0.9045
Epoch 49/50
225/225 ━━━━━━━━━━━━━━━━━━━━ 57s 255ms/step - accuracy: 0.8947 - loss: 0.3017 - val_accuracy: 0.7278 - val_loss: 1.0536
Epoch 50/50
225/225 ━━━━━━━━━━━━━━━━━━━━ 57s 255ms/step - accuracy: 0.8936 - loss: 0.2893 - val_accuracy: 0.7600 - val_loss: 0.9342
```

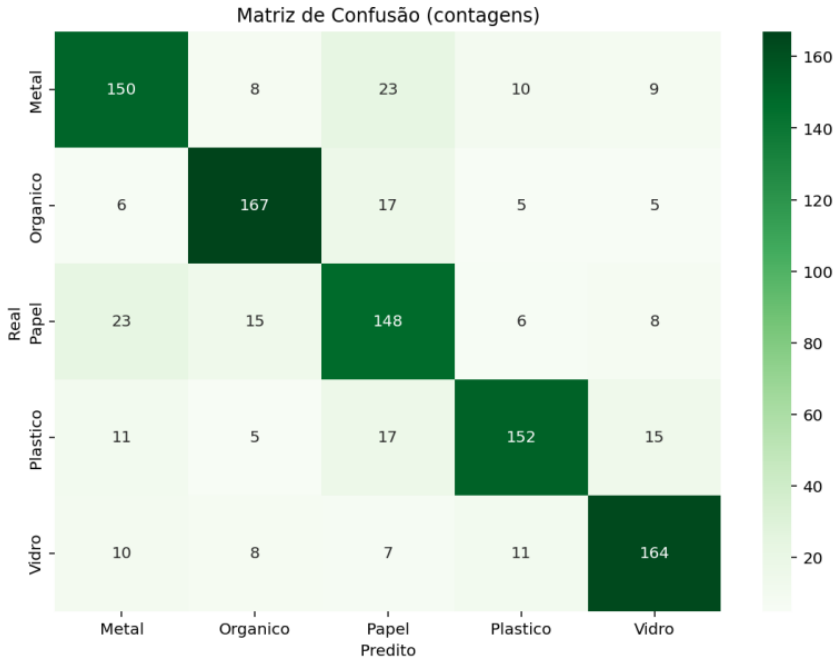




Teste:

	precision	recall	f1-score	support
Metal	0.75	0.75	0.75	200
Organico	0.82	0.83	0.83	200
Papel	0.70	0.74	0.72	200
Plastico	0.83	0.76	0.79	200
Vidro	0.82	0.82	0.82	200
accuracy			0.78	1000
macro avg	0.78	0.78	0.78	1000
weighted avg	0.78	0.78	0.78	1000

Especificidade média (macro): 0.9453
AUC média (macro): 0.9498



Conclusão

Esta segunda fase do projeto teve como objetivo otimizar a arquitetura da CNN da Fase I através de algoritmos de Inteligência Coletiva (Swarm Intelligence). O algoritmo escolhido, Gray Wolf Optimization (GWO), foi aplicado para encontrar a melhor combinação de três híper-parâmetros (Learning Rate, Dropout Rate, e N.º de Neurónios), sendo o seu desempenho comparado com o algoritmo base, o Particle Swarm Optimization (PSO).

Modelo	Híper-parâmetros	Accuracy (teste)	Auc (macro)
Meta 1	0.001 / 0.50 / 128	70.0%	0.9125
Meta 2 (PSO)	0.000954 / 0.52 / 162	76.0%	0.9395
Meta 2 (GWO)	0.000625 / 0.37 / 140	78.0%	0.9498

A Otimização Swarm foi Eficaz pois ambos os algoritmos de Swarm encontraram arquiteturas superiores à original. O modelo otimizado pelo PSO já representou um salto de 70% para 76%, enquanto o GWO conseguiu uma melhoria de 70% para 78%.

O algoritmo GWO provou ser a ferramenta de otimização mais eficaz para este problema. Não só encontrou um conjunto de híper-parâmetros que parecia mais promissor durante a fase de otimização (Custo GWO: 0.3422 | Custo PSO: 0.3672), como também resultou num modelo final com melhor desempenho em todas as métricas principais (Accuracy de 78% vs 76% e AUC de 0.9498 vs 0.9395).

O GWO identificou que um learning rate ligeiramente mais baixo (0.000625) e uma taxa de dropout significativamente menor (0.37) eram ideais para este modelo.

Embora os gráficos de treino ainda revelem overfitting, a capacidade de generalização do modelo no conjunto de teste melhorou substancialmente.