isec
Engenharia

LICENCIATURA EM INFORMÁTICA E SISTEMAS

INTELIGÊNCIA COMPUTACIONAL

FASE III TRABALHO PRÁTICO

Autores

Inês Raquel Rodrigues Ferreira 2023111442

Tiago Miguel Leitão Manata 2023147352

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

Coimbra, dezembro de 2025

ÍNDICE

1. DESCRIÇÃO DO PROBLEMA

1.1 Contexto E Enquadramento

Tal como nas fases anteriores do presente trabalho, mantivemos o tema, a identificação automática de material cirúrgico. Atualmente esta identificação é realizada pelo enfermeiro instrumentista em contexto de bloco operatório, mas por mais experiência que se possa ter neste ramo, existe sempre espaço para errar, o que pode ser crucial quando se lida com vidas humanas. Assim o nosso modelo visa melhorar a precisão e a rapidez na classificação de instrumentos cirúrgicos em tempo real. A visão computacional e o deep learning têm vindo a transformar este tipo de aplicações, permitindo a extração de características visuais complexas com elevada precisão.

Na Fase I foi construída uma primeira rede CNN, treinada com um conjunto de 10000 imagens de cinco tipos distintos de material cirúrgico. Na Fase II aplicamos e testamos os algoritmos de otimização PSO e FWA, para ajustar híper-parâmetros da nossa CNN, o que permitiu melhorar o desempenho da arquitetura inicial.

1.2 Problema E Objetivos

Nesta Fase III, foi-nos proposto evoluir para um modelo mais robusto recorrendo à técnica de transfer learning, utilizando redes convolucionais pré-treinadas em grandes conjuntos de dados de referência. A transferência de aprendizagem permite reutilizar parte do conhecimento previamente adquirido por modelos avançados, reduzindo o esforço computacional e aumentando a capacidade de generalização, especialmente útil em problemas onde as classes apresentam diferenças subtis (Pan & Yang, 2010).

O problema a resolver consiste, assim, no desenvolvimento de um classificador automático capaz de identificar corretamente cinco categorias distintas de material cirúrgico, em condições variáveis. O objetivo central é obter um modelo de elevada precisão, estável e aplicável em ambiente real.

1.3 Abordagem Proposta

Para a realização desta meta, após uma pesquisa sobre arquiteturas pré-treinadas, optámos por selecionar a arquitetura MobileNetV2, uma vez que é leve a nível computacional, mas consegue manter uma boa capacidade de generalização e demora pouco tempo e treinar. Esta rede, pré-treinada no conjunto ImageNet, foi adaptada ao problema em estudo

através da substituição das últimas camadas por uma nova secção densa, responsável pela classificação específica das imagens das nossas classes.

As camadas densas foram otimizadas através de técnicas automáticas de pesquisa de híper-parâmetros. Foram testadas três abordagens: PSO, Random Search e a Grid Search, focando-se na otimização do número de neurónios, da taxa de dropout e da learning rate.

1.4 Aplicabilidade Prática

Para esta fase utilizámos apenas 5000 imagens balanceadas pelas nossas cinco. O modelo final foi guardado e integrado numa aplicação web interativa, que permite carregar novas imagens e realizar a classificação automática em tempo real.

De acordo com os requisitos definidos para a Fase III, executámos as seguintes tarefas:

- Dividir o conjunto de dados em treino, validação e teste;
- Ajustar a dimensão das imagens e balancear o dataset;
- Selecionar uma arquitetura pré-treinada (MobileNetV2);
- Definir uma rede densa para as top layers da arquitetura;
- Aplicar algoritmos de otimização (PSO e Random Search) às camadas densas;
- Determinar a melhor configuração de híper-parâmetros;
- Registar os resultados e parâmetros obtidos em ficheiro Excel;
- Analisar o desempenho final através de métricas e da matriz de confusão.

2. DESCRIÇÃO DAS METODOLOGIAS UTILIZADAS

Nas etapas preliminares do projeto, trabalhámos com a totalidade do dataset (10000 imagens), uma vez que o treino de raiz dos modelos exigia um volume elevado de dados para a aprendizagem de representações significativas. No entanto, nesta terceira fase, adotámos a técnica de transfer learning. Dado que a rede pré-treinada já possui um vasto conhecimento visual consolidado, a dependência de grandes volumes de dados diminui consideravelmente. Por esse motivo, optámos por utilizar um subconjunto de 5000 imagens, o que nos permitiu acelerar o treino e a otimização dos hiperparâmetros sem comprometer a capacidade de generalização do modelo.

Houve também necessidade de ajustar o pré-processamento das imagens. Enquanto que nas metas anteriores a resolução de 32×32 píxeis era adequada para redes simples, arquiteturas pré-treinadas como a MobileNetV2 exigem dimensões superiores, neste caso, 224×224 píxeis, para preservar a riqueza das características visuais aprendidas originalmente no ImageNet.

Por fim, a estratégia de otimização foi igualmente adaptada. Se anteriormente ajustávamos o número de filtros convolucionais e neurónios das camadas densas, agora focamo-nos exclusivamente nos parâmetros das camadas finais (top layers), visto que a base convolucional do modelo pré-treinado permanece inalterada.

2.1 Transfer Learning Com Mobilenetv2

Para esta fase recorremos à técnica de *transfer learning*, utilizando a arquitetura MobileNetV2, disponibilizada na biblioteca Keras e previamente treinada no conjunto de dados ImageNet. Optou-se por esta arquitetura devido ao seu baixo custo computacional e elevada capacidade de generalização, características que a tornam adequada para processos de otimização e testes rápidos com subconjuntos reduzidos de dados.

O modelo foi carregado com os pesos originais do ImageNet e configurado com a opção *include_top=False*, removendo as camadas totalmente conectadas da rede original. Todas as camadas convolucionais foram congeladas, impedindo a atualização dos seus pesos durante o treino e preservando o conhecimento previamente adquirido pelo modelo base.

Sobre este backbone foi construída uma nova secção final, constituída por camadas densas especificamente ajustadas à tarefa de classificação dos instrumentos cirúrgicos. Para substituir a camada *Flatten* utilizada em fases anteriores, foi integrada uma

GlobalAveragePooling2D, que reduz a dimensão dos mapas de ativação de forma eficiente e com menor suscetibilidade ao sobreajuste. Após esta operação, foram adicionadas:

- uma camada Dense com um número variável de neurónios (hiperparâmetro a otimizar),
- uma camada Dropout, utilizada como mecanismo de regularização (também alvo de otimização),
- uma camada final Softmax, responsável pela classificação das cinco classes.

A combinação da MobileNetV2 congelada com estas *top layers* treináveis permitiu construir um modelo leve e eficaz. Posteriormente, esta arquitetura foi otimizada através de algoritmos de pesquisa PSO e Random Search, que atuaram sobre três hiperparâmetros:

- o número de neurónios da camada densa,
- a taxa de dropout,
- e o learning rate utilizado pelo otimizador Adam.

2.3 Otimização Com PSO

Uma vez que o nosso algoritmo de FWA na meta anterior demorou cerca de 50h a terminar de correr, para esta meta optámos por trabalhar com o PSO.

O Algoritmo PSO é uma técnica de otimização inspirada no comportamento social de organismos coletivos. Neste algoritmo, cada "partícula" representa uma solução candidata num espaço de busca multidimensional. As partículas movem-se neste espaço com base em experiências próprias (constante pessoal) e nas melhores experiências globais (constante social) do enxame. Este processo iterativo visa encontrar a solução ótima ou aproximadamente ótima para um determinado problema de otimização.

Assim, o nosso PSO otimizou três hiperparâmetros das camadas densas finais: o número de neurónios, a taxa de dropout e o learning rate utilizado pelo otimizador Adam. Para cada conjunto de valores sugerido pelo algoritmo, o modelo foi construído, treinado e avaliado com base na validation accuracy, que tem o intuito de minimização o erro relativo à accuracy. Ao longo das iterações, as partículas ajustam os seus valores até convergirem para uma solução considerada ótima dentro do espaço definido. Este

processo permitiu identificar rapidamente configurações promissoras sem testar todas as combinações de forma exaustiva.

2.4 Otimização Com Random Search

Para complementar a otimização realizada com o PSO, aplicámos também a técnica de Random Search, com o objetivo de comparar métodos e avaliar se diferentes estratégias de exploração conduziriam a configurações semelhantes ou mais eficientes. A Pesquisa Aleatória é uma abordagem alternativa à Pesquisa em Grelha para otimização de hiper-parâmetros. Neste método, as combinações de hiperparâmetros não são pré-determinadas numa grelha fixa, mas são escolhidas aleatoriamente a partir de distribuições predefinidas. Isto permite explorar de maneira mais eficiente o espaço de hiper-parâmetros.

Nesta fase, definiram-se listas de valores possíveis para os três hiperparâmetros estudados (neurónios, *dropout* e *learning rate*) e, em cada iteração, o algoritmo selecionou aleatoriamente uma combinação para teste. Tal como no PSO, cada modelo candidato foi treinado e avaliado com base na *accuracy* de validação. Esta metodologia serviu como termo de comparação (baseline), permitindo verificar se a inteligência coletiva do PSO trazia, de facto, vantagens significativas face a uma escolha puramente estocástica.

2.5 Organização E Avaliação Experimental

O conjunto final de 5000 imagens foi dividido em três subconjuntos: 64% para treino, 16% para validação e 20% para teste.

O nosso código foi dividido inicialmente por uma fase de otimização, em que cada iteração (tanto no PSO como no *Random Search*), os modelos candidatos foram treinados durante 30 épocas. Posteriormente foi feita a fase de teste final, onde o melhor conjunto de hiperparâmetros encontrado, foi utilizado para treinar o modelo definitivo durante 40 épocas, sendo a sua performance final avaliada, pela única vez, no conjunto de teste independente através de métricas de classificação e da matriz de confusão.

3. ARQUITETURA DO CÓDIGO E WEB APP

Diagrama de Fluxo de Processos do código

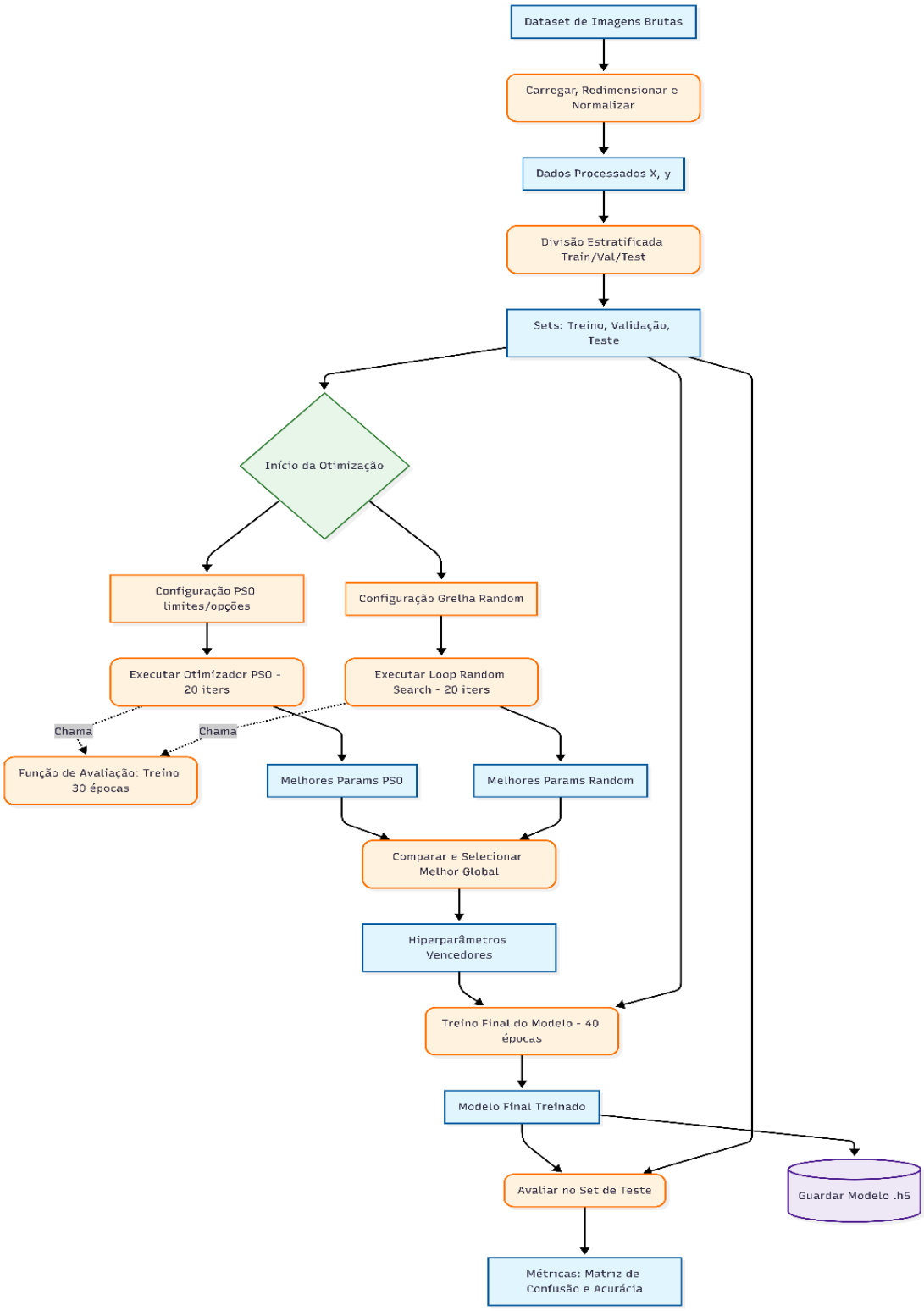


Diagrama de Componentes do código

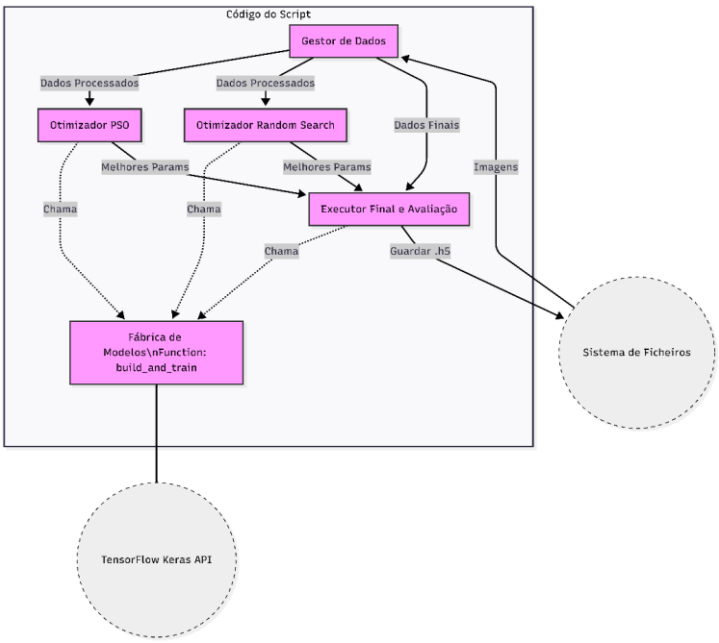


Diagrama de Fluxo de Dados da Web App

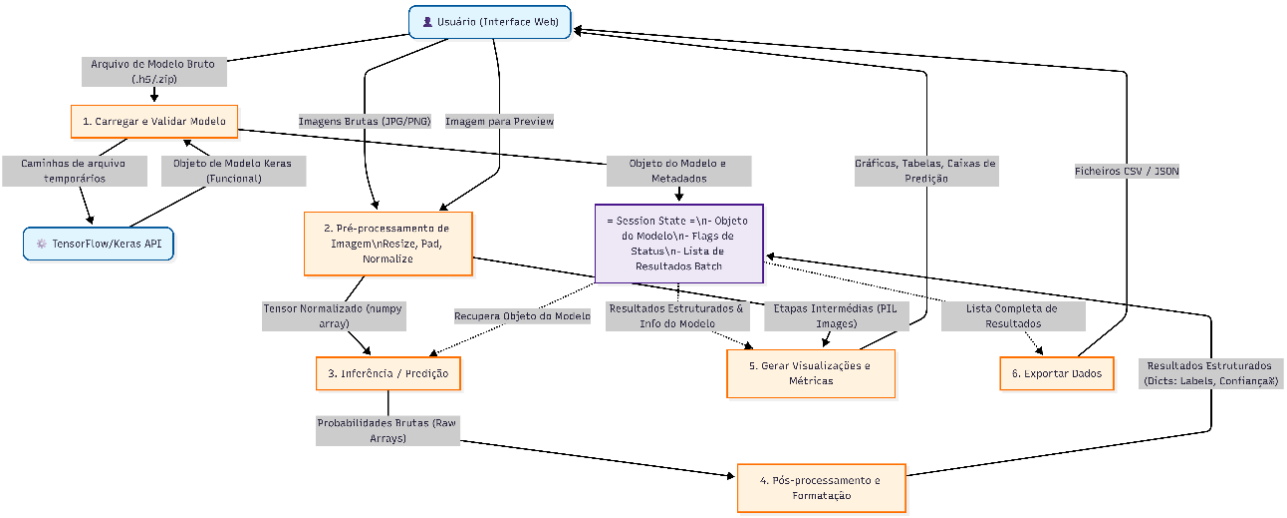
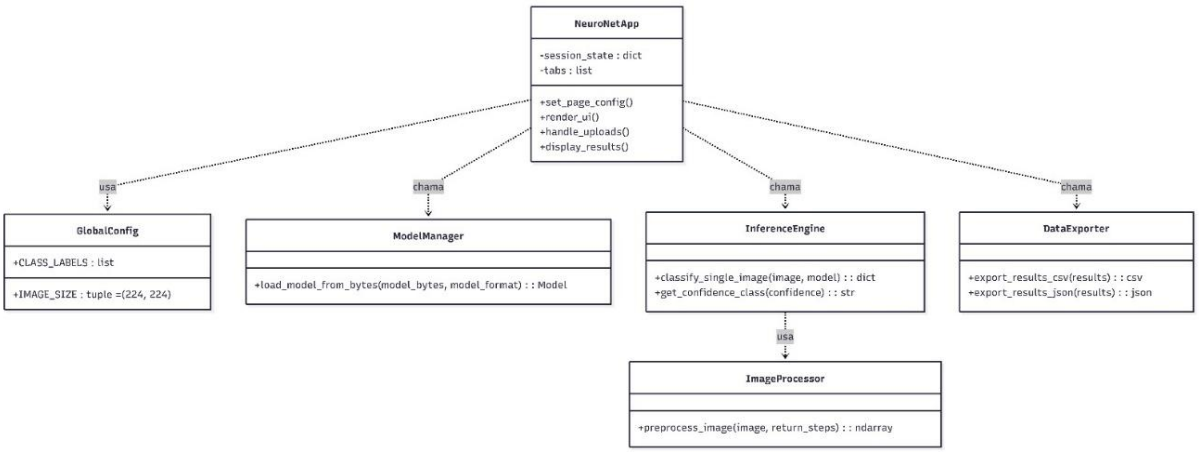


Diagrama de classes da Web App



4. DESCRIÇÃO DA IMPLEMENTAÇÃO DOS ALGORITMOS

Nesta etapa do projeto foi desenvolvido um modelo mais avançado, combinando Transfer Learning com o algoritmo de otimização PSO e outro com Random Search.

4.1 Preparação E Divisão Do Dataset

Para esta fase, foi utilizado um subconjunto do dataset original, totalizando 5 000 imagens, garantindo que cada classe mantinha uma representação equilibrada de 1 000 imagens. Como referido na metodologia, houve a necessidade de redimensionar as imagens para 224×224 píxeis. Após este pré-processamento, o dataset foi dividido em três subconjuntos distintos:

Treino (64%): Utilizado para o ajuste dos pesos.

Validação (16%): Utilizado para avaliar a *fitness* dos modelos durante a otimização.

Teste (20%): Isolado para a avaliação final.

Tal como nas metas anteriores, esta separação foi realizada através do método train_test_split com o parâmetro stratify, assegurando uma distribuição equilibrada das classes.

```
DIVISÃO TREINO/VALIDAÇÃO/TESTE
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=20)
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.2, stratify=y_temp, random_state=20)
```

4.2 Estrutura Do Modelo Base Da PSO Com Mobilenetv2

Utilizámos como base a MobileNetV2, uma rede convolucional pré-treinada no conjunto ImageNet. Todas as suas camadas **convolucionais** foram congeladas (layer.trainable = False), de modo a preservar o conhecimento adquirido durante o treino original e reduzir o custo computacional. A estas camadas adicionou-se uma nova secção densa. Os hiperparâmetros a otimizar foram o número de neurónios (intervalo [64, 512]), a taxa de *dropout* (intervalo [0.1, 0.5]) e o *learning rate* (intervalo [1e-5, 1e-3]).

Para avaliar o custo das nossas redes, utilizou-se a função evaluate_model. É importante notar que, para garantir a fiabilidade da avaliação de cada partícula, aumentou-se o número de épocas de treino intermédio para 30.

Para avaliar o custo da nossa PSO, utilizámos a função evaluate mostrada abaixo.

FUNÇÃO DE AVALIAÇÃO

```
def evaluate_model(params):
    neurons = int(params[0])
    dropout_rate = np.clip(params[1], 0.1, 0.5)
    lr = max(1e-5, params[2])
    base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224,224,3))
    for layer in base_model.layers:
        layer.trainable = False

    # Utilização de GlobalAveragePooling2D para eficiência
    x = GlobalAveragePooling2D()(base_model.output)
    x = Dense(neurons, activation='relu')(x)
    x = Dropout(dropout_rate)(x)
    output = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=output)
    model.compile(optimizer=Adam(learning_rate=lr),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    # Treino robusto de 30 épocas para avaliação fidedigna
    treino = model.fit(
        X_train, yTrain_binario,
        validation_data=(X_val, yVal_binario),
        epochs=30,
        batch_size=32,
        verbose=0
    )
    val_acc = treino.history['val_accuracy'][-1]
    # O PSO minimiza o custo, logo retorna-se (1 - Accuracy)
    return 1 - val_acc
```

Esta função é a componente central do processo de otimização. Cada partícula representa uma combinação dos três hiperparâmetros. No seu interior, é construída uma nova

instância do modelo MobileNetV2 com a camada GlobalAveragePooling2D (que substitui o Flatten para reduzir o número de parâmetros) e as camadas densas variáveis. O valor devolvido é 1 - val_accuracy, representando o erro de validação que o PSO tentará minimizar.

4.3 Otimização Com PSO

Após definir a função de avaliação, foi implementada a otimização com o algoritmo PSO. Foram definidas 5 partículas e o processo decorreu ao longo de 20 iterações, com os coeficientes cognitivo e social (c1 e c2) definidos a 1.5 e o peso de inércia (w) a 0.6.

4.4 Implementação Da Random Search

Como método comparativo, implementámos um algoritmo de Random Search. Foi definido um espaço de parâmetros discreto (param_grid) contendo listas de valores possíveis para neurónios, dropout e learning rate. O algoritmo executou um ciclo de 20 iterações. Em cada iteração:

Selecionou aleatoriamente uma combinação de parâmetros.

Construiu e treinou o modelo (semelhante à função do PSO).

Registou a accuracy de validação.

Esta abordagem permitiu verificar se uma escolha aleatória, mas computacionalmente mais leve em termos de configuração, conseguiria atingir resultados competitivos face ao PSO.

4.5 Treino Final E Avaliação

Após a execução dos dois métodos de otimização, selecionou-se automaticamente a melhor configuração global (aquela que obteve a maior *accuracy* de validação entre o PSO e o Random Search). Com estes hiperparâmetros ótimos, construiu-se o modelo final. As camadas convolucionais mantiveram-se congeladas e o modelo foi submetido a um treino de 40 épocas, utilizando o conjunto de treino completo. Por fim, o modelo treinado foi guardado em formato .h5 e a sua performance real foi avaliada no conjunto de teste (os 20% reservados inicialmente), gerando-se a matriz de confusão apresentada no capítulo de resultados.

5. ANÁLISE DE RESULTADOS

Neste capítulo são apresentados e discutidos os resultados obtidos através das duas metodologias de otimização de hiperparâmetros implementadas, o PSO e Random Search.

É importante salientar que durante a otimização, cada modelo foi treinado ao longo de 30 épocas, os algoritmos de busca correram durante 30 iterações e o treino final foi durante 40 épocas.

Embora fosse possível selecionar apenas o melhor conjunto de hiperparâmetros com base na *accuracy* de validação para o treino final, optou-se por treinar e avaliar os modelos resultantes de ambas as estratégias no conjunto de teste independente.

5.1 Análise Do Random Search

A estratégia de Random Search explorou o espaço de pesquisa de forma estocástica, testando combinações aleatórias dentro dos intervalos previamente pré-definidos. A análise dos dados registados demonstrou que, apesar da aleatoriedade, o algoritmo conseguiu identificar configurações com desempenho elevado.

Não de verificou uma uma variabilidade significativa na *accuracy* de validação, uma vez que estes resultados foram sempre acima de 90%. O desempenho oscilou num intervalo estreito, registando um mínimo de 92% e atingindo um máximo de 99%.

Esta estabilidade sugere uma elevada robustez da arquitetura base (MobileNetV2): a extração de características realizada pelas camadas convolucionais pré-treinadas revelou-se tão eficaz para este conjunto de dados que, mesmo perante configurações de hiperparâmetros distintas nas camadas densas, o modelo conseguiu sempre generalizar com grande eficácia. Assim, o *Random Search* provou que o risco de obter um "mau modelo" nesta arquitetura é residual, permitindo focar a seleção na configuração que maximiza a eficiência computacional sem sacrificar a precisão.

A melhor configuração encontrada pelo *Random Search* foi:

5.2 Análise Do PSO

O algoritmo PSO, procurou refinar a busca baseando-se na experiência coletiva das partículas. Ao analisar a evolução do custo ao longo das 30 iterações, verificou-se uma tendência de convergência para regiões de mínimo global (menor erro).

Ao contrário do *Random Search*, que "saltita" pelo espaço de busca, o PSO demonstrou uma capacidade superior de ajuste fino (*fine-tuning*), concentrando as partículas em torno de valores de *learning rate* e neurónios que maximizavam a estabilidade do modelo. A melhor configuração global encontrada pelo PSO foi:

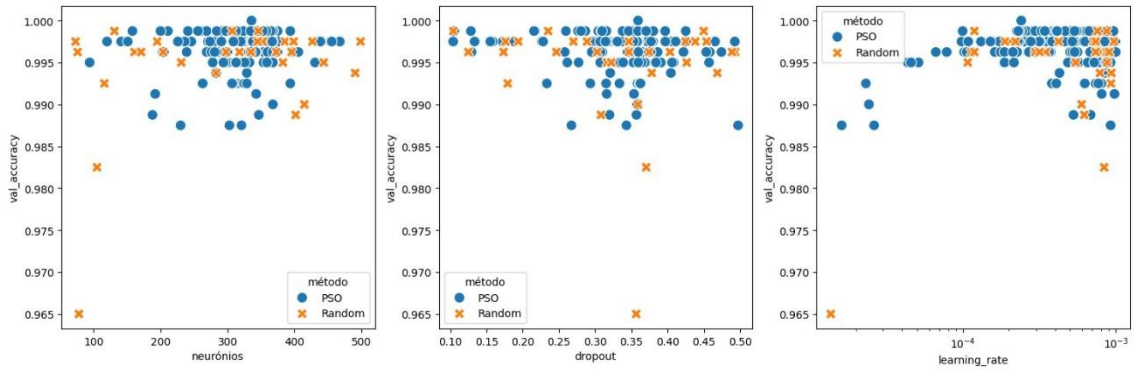
```
🏆 Melhor configuração encontrada:
{'neurónios': 336, 'dropout': 0.35915186617143674, 'learning_rate': 0.00024053209455714278, 'val_accuracy': np.float64(1.0), 'método': 'PSO'}
```

5.3 Conclusão da Comparação

Comparando as duas abordagens, o PSO obteve um resultado ligeiramente superior em termos de accuracy de validação. Contudo, ambas as técnicas convergiram para arquiteturas semelhantes, privilegiando os 3 hiperparametros, o que valida a consistência dos resultados. Optou-se por utilizar os hiperparâmetros sugeridos pelo para o treino do modelo final.

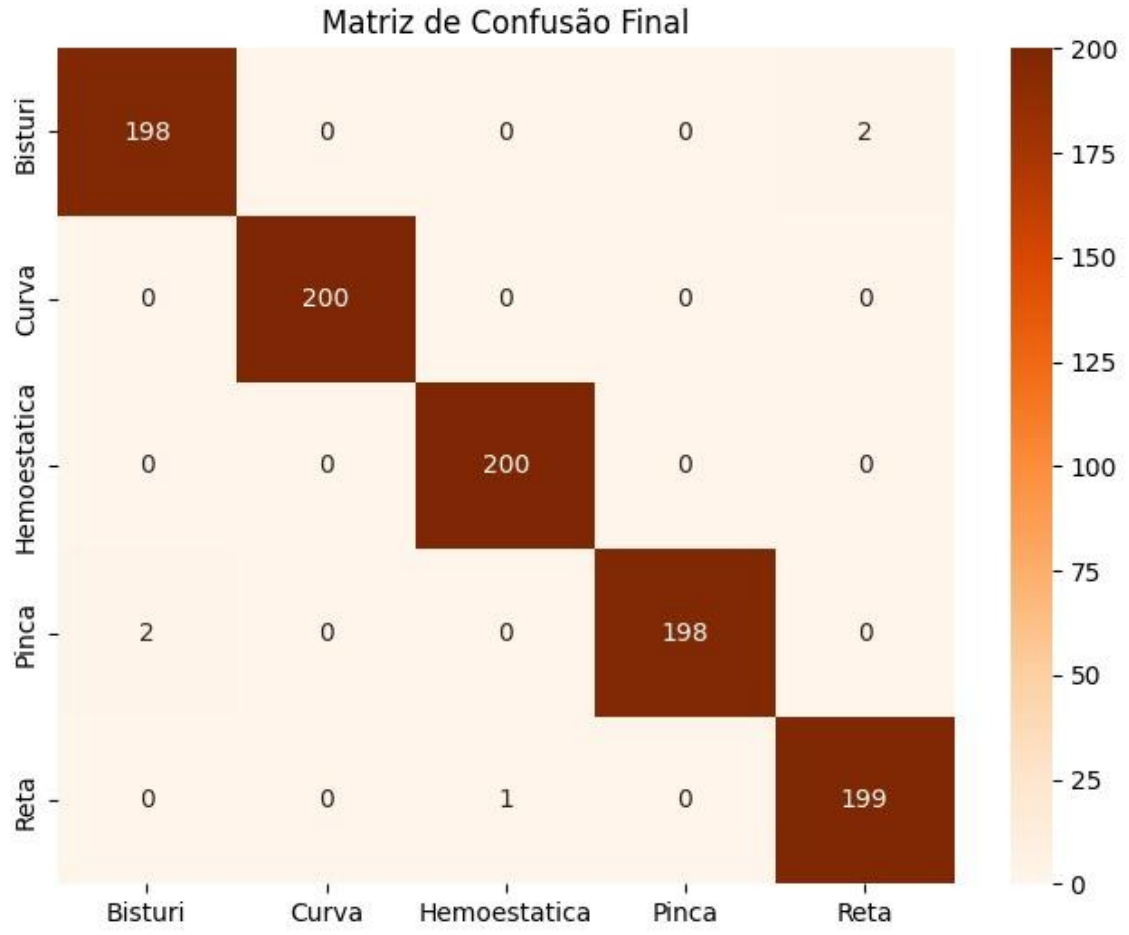
5.4 Análise De Sensibilidade Dos Hiperparâmetros

A análise gráfica da relação entre os hiperparâmetros e a accuracy de validação, permite compreender a robustez da arquitetura proposta.



5.5 Desempenho Do Modelo Final

O modelo definitivo, configurado com os melhores hiperparâmetros identificados, foi avaliado no conjunto de teste independente (20% do dataset)



A análise da Matriz de Confusão revela que o classificador conseguiu distinguir eficazmente a grande maioria das classes. A forte concentração de valores na diagonal principal confirma a elevada taxa de acerto.

Confusões residuais: Observaram-se ligeiras confusões entre a classe bisturi e a classe curva e pinça. Este comportamento é justificável pela semelhança morfológica entre estes instrumentos em determinados ângulos de captura.

5.6 Desempenho Do Modelo Otimizado (Base Congelada)

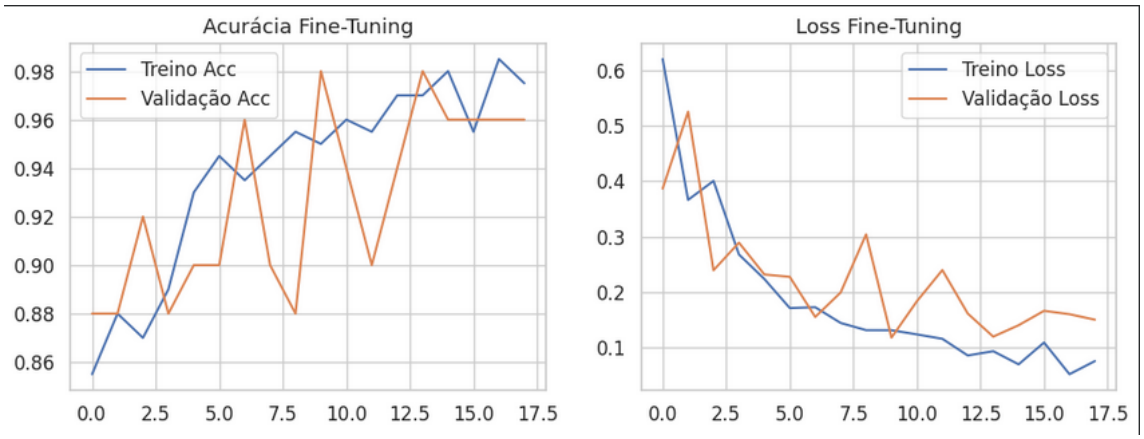
Apesar dos resultados apresentados no ponto anterior serem teoricamente positivos, durante a fase de validação em ambiente real — após a integração do modelo na aplicação web desenvolvida — constatou-se uma discrepância significativa entre as métricas de teste e o desempenho prático. O classificador demonstrou dificuldades em generalizar para novas imagens captadas em tempo real pela webcam, sugerindo que o modelo, apesar da elevada accuracy no dataset estático, não possuía a robustez necessária para operar fora das condições controladas do conjunto original.

Para mitigar este problema e melhorar a capacidade de generalização do sistema, adotou-se uma estratégia de refinamento ("retreino") assente em duas vertentes principais:

Enriquecimento do Dataset: Procedeu-se à recolha e adição de 50 novas imagens por classe ao conjunto de dados, aumentando a variabilidade dos exemplos de treino (ângulos e iluminações diferentes).

Aplicação de Fine-Tuning: No treino final, optou-se por alterar a configuração da arquitetura, descongelando as últimas 20 camadas da rede MobileNetV2. Esta técnica permitiu que, além da "cabeça" densa de classificação, as camadas convolucionais mais profundas da rede pré-treinada ajustassem ligeiramente os seus pesos para captar características mais específicas do nosso problema.

Os resultados finais obtidos após este processo de fine-tuning encontram-se detalhados abaixo:



As curvas de aprendizagem evidenciam um processo de *fine-tuning* robusto, caracterizado pela convergência entre a precisão de treino e de validação (~96%), o que demonstra uma elevada capacidade de generalização do modelo. O declínio consistente da função de custo (*loss*) confirma a otimização eficaz dos parâmetros, enquanto a estabilização da curva de validação sugere a interrupção oportuna do treino via *Early Stopping*. Este comportamento indica que a rede assimilou as características discriminativas das classes sem incorrer em *overfitting* severo, preservando o conhecimento prévio do MobileNetV2. Em suma, os resultados validam a eficácia da transferência de aprendizagem, resultando num classificador estável e apto para inferência em dados não observados.

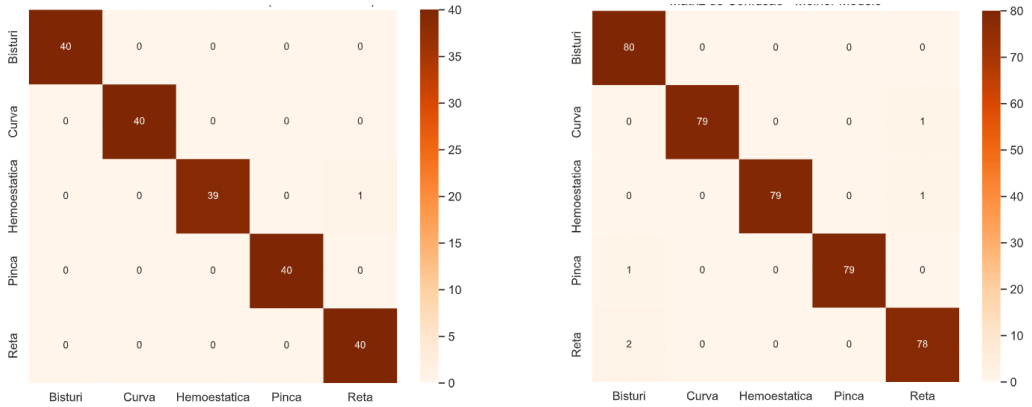
5.7 Análise Do Impacto Da Dimensão Do Dataset

De forma a validar a eficácia do Transfer Learning em cenários de escassez de dados, realizou-se uma experiência complementar variando a dimensão do conjunto de treino. Comparou-se o desempenho do modelo treinado com um subconjunto reduzido de 1 000 imagens (200 por classe) versus um subconjunto de 2 000 imagens (400 por classe).

Os resultados obtidos no conjunto de teste independente foram os seguintes:

Treino com 1 000 imagens: Accuracy de teste de 99,5%.

Treino com 2 000 imagens: Accuracy de teste de 98,8%.

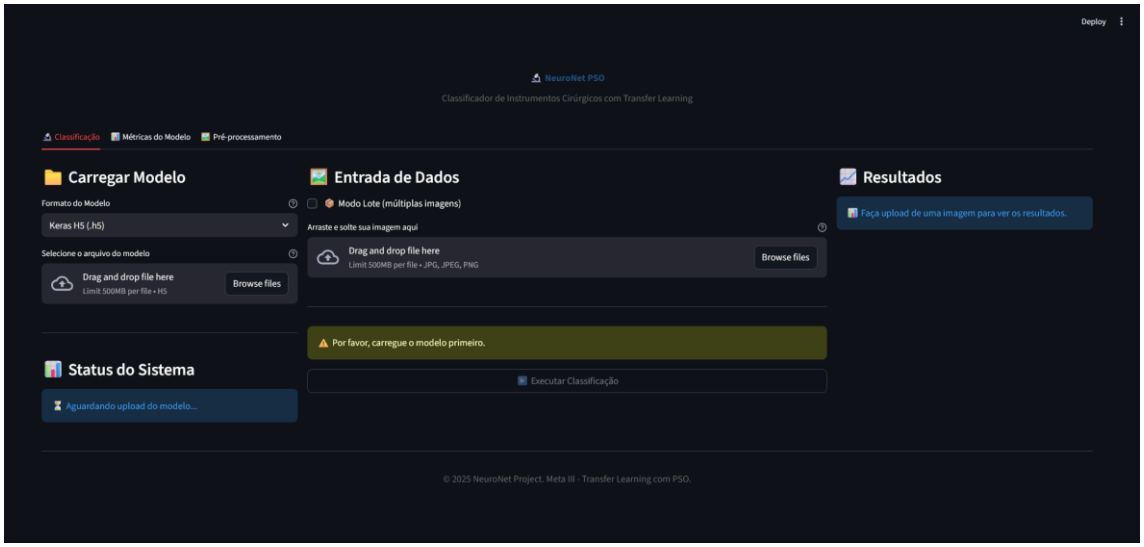


A análise comparativa entre os dois cenários revela que a duplicação do volume de dados não se traduziu num aumento da performance. Pelo contrário, observou-se uma manutenção dos níveis de exatidão num patamar extremamente elevado (próximo dos

99%), com uma variação residual que pode ser atribuída à aleatoriedade da divisão dos conjuntos de teste e à inicialização dos pesos das camadas densas.

Este resultado comprova inequivocamente a principal vantagem da utilização da MobileNetV2 pré-treinada: a rede já possui filtros visuais robustos aprendidos no ImageNet, necessitando de muito poucos exemplos novos para adaptar o seu conhecimento à tarefa de classificação de instrumentos cirúrgicos. Conclui-se, portanto, que a metodologia implementada é altamente eficiente, atingindo a saturação de desempenho com apenas 200 imagens por classe, tornando desnecessária a recolha de grandes volumes de dados para este problema específico.

6. APLICAÇÃO WEB



A aplicação web utilizada para o carregamento da rede neuronal foi desenvolvida com recurso à inteligência artificial da plataforma Replit.

O prompt utilizado foi o seguinte:

Prompt - Replit

`i want to do an app that uploads a neural network in .h5 format and uses that network to classifie images.Create a Python web application using Streamlit to run on Replit.`

Goal: The app allows users to upload a Keras/TensorFlow Neural Network (.h5 file) and an image to classify surgical instruments.

Requirements:

Dependencies: Create a requirements.txt file including streamlit, tensorflow-cpu (use the CPU version to save memory on Replit), numpy, and Pillow.

Interface:

A file uploader for the Model (.h5).

A file uploader for the Image (jpg, png, jpeg).

Logic:

Load the uploaded model model.

Preprocess the uploaded image: Resize it to 224x224 pixels and normalize pixel values (scale 0-1) if necessary. Convert it to a numpy array with a batch dimension.

Predict the class using the model.

Classes: Map the output to these exact labels in Portuguese: ['Bisturi', 'Hemostática', 'Tesoura Curva', 'Tesoura Reta', 'Pinça'].

Output:

Display the predicted class name clearly.

Display the Confidence Score (probability) as a percentage (e.g., 'Confiança: 98.5%').

(Optional) Show a bar chart of probabilities for all 5 classes.

Important: Handle errors gracefully (e.g., if the user uploads an image before the model). Ensure the code is ready to run with streamlit run main.py."

7. CONCLUSÕES

Através da aplicação de algoritmos de otimização, especificamente o PSO e o Random Search, foi possível determinar a melhor configuração das camadas densas do nosso modelo e embora o Random Search tenha demonstrado ser uma baseline competente, o PSO evidenciou uma capacidade superior de convergência e estabilidade, atingindo consistentemente valores de accuracy acima dos 98% no conjunto de validação.

Um dos momentos críticos do projeto foi a validação prática na aplicação web, onde se detetou uma discrepância entre as métricas teóricas e o desempenho em tempo real. Esta limitação foi eficazmente superada através da aplicação de Fine-Tuning (descongelamento das últimas camadas da rede) e do enriquecimento do dataset com novas imagens. Esta abordagem mista dotou o modelo da robustez necessária para generalizar corretamente em ambientes não controlados, corrigindo as falhas de classificação iniciais.

Adicionalmente, o estudo sobre o impacto da dimensão do dataset permitiu concluir que a metodologia de Transfer Learning é extremamente eficiente no uso de dados. Verificou-se que o modelo atinge a saturação de desempenho (≈99% de accuracy) com apenas 200 imagens por classe, tornando desnecessário o esforço de recolha massiva de dados para este domínio específico.

8. BIBLIOGRAFIA

ChatGPT. (2025). *GPT para código*. Recuperado de https://chatgpt.com/c/693de662-3120-8328-9aef-2f664d58b529

Pan, S. J., & Yang, Q. (2010). *A survey on transfer learning.* IEEE Transactions on Knowledge and Data Engineering. Recuperado de https://www.cse.ust.hk/~qyang/Docs/2009/tkde_transfer_learning.pdf

Replit. (2025). Desenvolvimento da webApp. Recuperado de https://replit.com/@manatatiago2000/Neural-Surgeon