Relatório Projeto – Meta III

Transfer Learning

Inteligência Computacional

Lucas Pagnano Golobovante - 2023140728
Diogo Valdemar Melo Estimado - 2023130197

Índice

Lucas Pagnano Golobovante – 2023140728
Diogo Valdemar Melo Estimado - 2023130197

Descrição do Problema

Introdução

O presente projeto insere-se na Fase III da unidade curricular de Inteligência Computacional, tendo como objetivo principal a implementação de um sistema robusto de classificação de imagens utilizando técnicas de Aprendizagem por Transferência. Enquanto nas fases anteriores o foco residiu na construção e treino de redes neuronais "do zero" e na otimização básica, esta etapa pretende combater as limitações de recursos computacionais e a necessidade de grandes volumes de dados.

O problema central abordado é a classificação automática de resíduos, uma tarefa complexa devido à variabilidade visual dos objetos. O desafio técnico desta fase consiste em reutilizar o conhecimento de modelos pré-treinados em grandes bases de dados (ImageNet) para atingir elevada precisão no nosso problema específico, mesmo com um conjunto de dados reduzido.

Objetivo

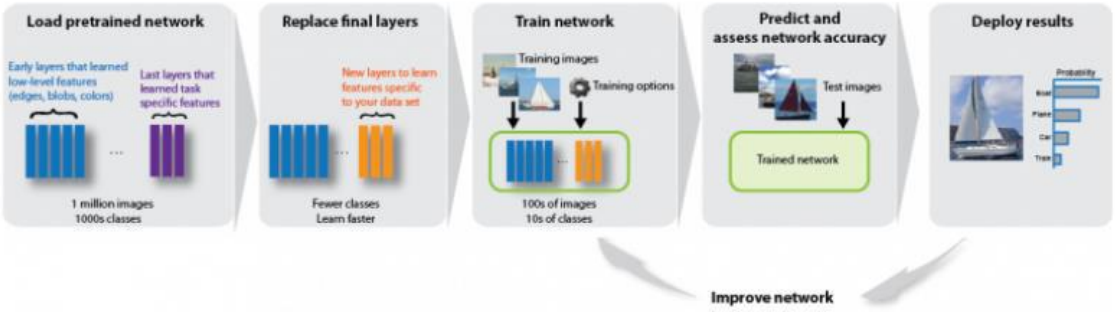
- Implementar uma arquitetura baseada na MobileNetV2 utilizando Transfer Learning.
- Otimizar os híper-parâmetros das camadas densas utilizando o algoritmo Grey Wolf Optimizer e comparar o seu desempenho com uma pesquisa aleatória (Random Search).
- Avaliar a robustez do modelo treinando-o com frações reduzidas do conjunto de dados (100%, 50% e 25%).
- Implementar o modelo final numa aplicação Web para classificação em tempo real.

Metodologias Utilizadas

Aprendizagem por Transferência

A Aprendizagem por Transferência é uma técnica de Deep Learning onde um modelo desenvolvido para uma tarefa é reutilizado como ponto de partida para um modelo numa segunda tarefa. Em vez de iniciar o treino com pesos aleatórios, utilizam-se os pesos pré-treinados que já aprenderam a extrair características visuais complexas (bordas, texturas, formas).

A metodologia adotada consistiu em utilizar a rede MobileNetV2 para extrair características. A base convolucional da rede foi "congelada" (frozen), impedindo a atualização dos seus pesos, enquanto novas camadas densas foram adicionadas e treinadas especificamente para as classes do nosso projeto



Arquitetura MobileNetV2

A escolha da MobileNetV2 justifica-se pela sua eficiência e leveza, sendo ideal para a implementação em ambientes com recursos limitados ou aplicações web. Este modelo, treinado na base de dados ImageNet (com mais de um milhão de imagens), utiliza convoluções separáveis em profundidade (depthwise separable convolutions) que reduzem drasticamente o número de parâmetros e o custo computacional sem sacrificar significativamente a precisão.

Algoritmos de Otimização

Para definir a configuração ideal das novas camadas densas, foram comparados dois métodos de otimização:

- Grey Wolf Optimizer (GWO): Um algoritmo de inteligência de enxame inspirado na hierarquia social e estratégia de caça dos lobos cinzentos. O algoritmo simula a liderança dos lobos Alfa, Beta e Delta para guiar a população em direção à solução ótima global no espaço de pesquisa.
- Random Search: Um método estocástico que seleciona combinações aleatórias de híper-parâmetros. Foi utilizado como baseline para validar se a procura guiada do GWO oferece vantagens reais de desempenho em comparação com a aleatoriedade pura.

Os híper-parâmetros otimizados foram: o número de neurónios na camada densa, a taxa de Dropout e a taxa de aprendizagem (Learning Rate).

Pré-processamento de Dados

Ao contrário da normalização simples (1./255) utilizada em fases anteriores, a MobileNetV2 requer um pré-processamento específico que ajusta os valores dos pixéis para o intervalo [-1, 1]. Foi utilizada a função preprocess_input do Keras para garantir que as estatísticas das imagens de entrada coincidem com as que o modelo utilizou durante o seu treino original na ImageNet.

Apresentação da Arquitetura de Código e Web App

Arquitetura do Sistema

A solução foi desenvolvida de forma modular para garantir a separação de responsabilidades entre a definição do modelo, a otimização de híper-parâmetros e a interface final de utilização.

Estrutura dos Ficheiros e Componentes

setup_transfer.py: Módulo responsável pela definição da arquitetura da Rede Neuronal. Contém a função build_transfer_model, que carrega a base MobileNetV2 e acopla as camadas densas personalizadas.

otimizar_transfer.py: Script que implementa a lógica de procura dos melhores híper-parâmetros utilizando a biblioteca SwarmPackagePy (GWO) e o algoritmo Random Search.

treino.py: Script de validação que treina o modelo com os parâmetros vencedores em diferentes frações do dataset e gera os ficheiros .h5 finais.

teste.py: Script para testar o modelo treinado e gerar um report e uma matriz de confusão.

app.py: Aplicação Web desenvolvida em Streamlit, que carrega o modelo treinado e fornece uma interface gráfica para classificação de imagens em tempo real.

Bibliotecas principais

TensorFlow/Keras: Para construção e treino da rede neuronal e implementação do Transfer Learning.

SwarmPackagePy: Para a implementação do algoritmo de otimização Grey Wolf Optimizer.

Pandas/Numpy: Para manipulação de dados e registo de logs de treino.

Streamlit: Para o desenvolvimento rápido da interface Web.

Lucas Pagnano Golobovante – 2023140728
Diogo Valdemar Melo Estimado - 2023130197

Descrição da Implementação dos Algoritmos

Implementação do Modelo de Transfer Learning

A construção do modelo foi implementada através da função build_transfer_model. A implementação segue rigorosamente a metodologia de Feature Extraction:

1. A instância da MobileNetV2 é carregada com os pesos da 'imagenet' e com o argumento include_top=False, removendo a camada de classificação original de 1000 classes.
2. A propriedade trainable é definida como False, "congelando" os pesos convolucionais para preservar as características aprendidas.
3. São adicionadas camadas GlobalAveragePooling2D, seguidas de uma camada Dense e uma camada de Dropout.

```python
def build_transfer_model(learning_rate, dropout_rate, dense_neurons, num_classes):
    # 1. Carregar a "Base" (O cérebro pré-treinado)
    base_model = MobileNetV2(
        weights='imagenet',  # Usar conhecimento prévio [cite: 1091]
        include_top=False,   # Não incluir a camada final de classificação (1000 classes)
        input_shape=(224, 224, 3)
    )

    # 2. Congelar a base (Não queremos estragar o que a rede já sabe)
    base_model.trainable = False

    # 3. Criar a nova "Cabeça" (Top Layers)
    model = models.Sequential()
    model.add(base_model)
    model.add(layers.GlobalAveragePooling2D()) # Resume a informação da base
    model.add(layers.Dense(dense_neurons, activation='relu')) # Camada densa que vamos otimizar
    model.add(layers.Dropout(dropout_rate)) # Dropout que vamos otimizar
    model.add(layers.Dense(num_classes, activation='softmax')) # 5 Classes de lixo

    # 4. Compilar
    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

Algoritmo de Otimização GWO

A otimização dos híper-parâmetros foi implementada integrando a classe GWO da biblioteca SwarmPackagePy com uma função de fitness personalizada.

A função de fitness recebe um vetor de posições dos "lobos" (agentes) correspondente a [Neurónios, Dropout, Learning Rate]. Para cada iteração:

1. Constrói-se um novo modelo com esses parâmetros.
2. Treina-se o modelo por um número reduzido de épocas.
3. Retorna-se o valor 1 - Val_Accuracy como custo a minimizar.

Este processo permitiu explorar o espaço de pesquisa de forma guiada, convergindo para uma combinação de parâmetros mais eficiente do que a escolha manual.

```python
def fitness_function(params):
    """
    Função objetivo a ser MINIMIZADA pelo GWO.
    Retorna (1 - val_accuracy).
    """
    # 1. Descodificar Parâmetros
    n_neurons = int(params[0])   # Cast para int
    dropout_rate = params[1]
    learning_rate = params[2]

    # Garantir limites de segurança caso o algoritmo passe um pouco
    n_neurons = max(int(LB[0]), min(int(UB[0]), n_neurons))
    dropout_rate = max(LB[1], min(UB[1], dropout_rate))
    learning_rate = max(LB[2], min(UB[2], learning_rate))

    # Limpeza de sessão para evitar erro de memória
    tf.keras.backend.clear_session()

    # 2. Construir Modelo MobileNetV2
    base_model = MobileNetV2(weights='imagenet',
                             include_top=False,
                             input_shape=(224, 224, 3))
    base_model.trainable = False # Congelar pesos base

    x = base_model.output
    x = GlobalAveragePooling2D()(x)

    # Camadas a Otimizar
    x = Dense(n_neurons, activation='relu')(x)
    x = Dropout(dropout_rate)(x)

    # Camada Final
    # No teu caso são 5 classes, então categorical_crossentropy e softmax
    num_classes = 5
    activation = 'softmax'
    loss_type = 'categorical_crossentropy'

    outputs = Dense(num_classes, activation=activation)(x)
    model = Model(inputs=base_model.input, outputs=outputs)

    # 3. Compilar e Treinar
    model.compile(optimizer=Adam(learning_rate=learning_rate),
                  loss=loss_type,
                  metrics=['accuracy'])

    # Treino curto
    history = model.fit(train_generator,
                        validation_data=val_generator,
                        epochs=EPOCHS,
                        verbose=0) # Silencioso na consola

    # 4. Calcular Fitness
    best_val_acc = max(history.history['val_accuracy'])
    fitness_val = 1.0 - best_val_acc # Minimização

    # 5. Guardar no Log (Excel)
    # Apanhamos os dados "em tempo real"
    results_log.append({
        'Algorithm': CURRENT_ALGO,
        'Neurons': n_neurons,
        'Dropout': dropout_rate,
        'Learning_Rate': learning_rate,
        'Accuracy': best_val_acc,
        'Fitness (1-Acc)': fitness_val
    })

    # Print de progresso
    print(f"[{CURRENT_ALGO}] Neurons={n_neurons}, Drop={dropout_rate:.2f}, LR={learning_rate:.5f} -> Acc={best_val_acc:.4f}")

    return fitness_val
```

```python
print("\n--- A INICIAR GWO (SwarmPackagePy) ---")
CURRENT_ALGO = "GWO"

# A biblioteca usa: gwo(n, function, lb, ub, dimension, iteration)
gwo_algo = gwo(N_AGENTS, fitness_function, LB, UB, 3, N_ITER)

best_gwo_params = gwo_algo._sw__Gbest
print(f"Melhor GWO: {best_gwo_params}")
```

Lucas Pagnano Golobovante – 2023140728
Diogo Valdemar Melo Estimado - 2023130197

Implementação Random Search

Para validar a eficácia do algoritmo GWO, foi implementado um algoritmo de Random Search para servir como termo de comparação. A implementação deste algoritmo foi desenhada para garantir condições de teste rigorosamente idênticas às do GWO:

1. Espaço de Procura: O algoritmo gera coordenadas aleatórias respeitando os mesmos limites inferiores e superiores (LB e UB) definidos para o GWO.
2. Função de Custo: Ambos os algoritmos invocam exatamente a mesma fitness_function. Isto garante que o treino provisório e a avaliação da Accuracy são realizados da mesma forma, eliminando variáveis externas.
3. Registo de Dados: Tal como no GWO, cada iteração do Random Search regista os híper-parâmetros testados e a precisão obtida.

Esta abordagem permite verificar se a "inteligência" do enxame do GWO consegue efetivamente encontrar melhores soluções do que uma pesquisa puramente estocástica dentro do mesmo espaço de possibilidades.

```
print("\n--- A INICIAR RANDOM SEARCH ---")
CURRENT_ALGO = "Random Search"

# Para ser justo, fazemos o mesmo número de avaliações que o GWO
total_evals = N_AGENTS * N_ITER

best_rs_fitness = 1.0
best_rs_params = []

for i in range(total_evals):
    # Gerar parâmetros aleatórios
    r_neurons = np.random.randint(LB[0], UB[0] + 1)
    r_dropout = np.random.uniform(LB[1], UB[1])
    r_lr = np.random.uniform(LB[2], UB[2])

    params = [r_neurons, r_dropout, r_lr]

    # Chamar a mesma função de fitness
    fit = fitness_function(params)

    if fit < best_rs_fitness:
        best_rs_fitness = fit
        best_rs_params = params

print(f"Melhor Random Search: {best_rs_params} | Fitness: {best_rs_fitness}")
```

Análise da Redução de Dados

Para cumprir o requisito de avaliação do impacto do número de instâncias, foi implementado um script dedicado (treino.py). Este algoritmo utiliza a divisão estratificada dos dados para criar subconjuntos de treino contendo 100%, 50% e 25% das imagens originais, mantendo o conjunto de validação fixo para garantir a comparabilidade justa dos resultados.

Análise dos Resultados

Otimização de Híper-parametros

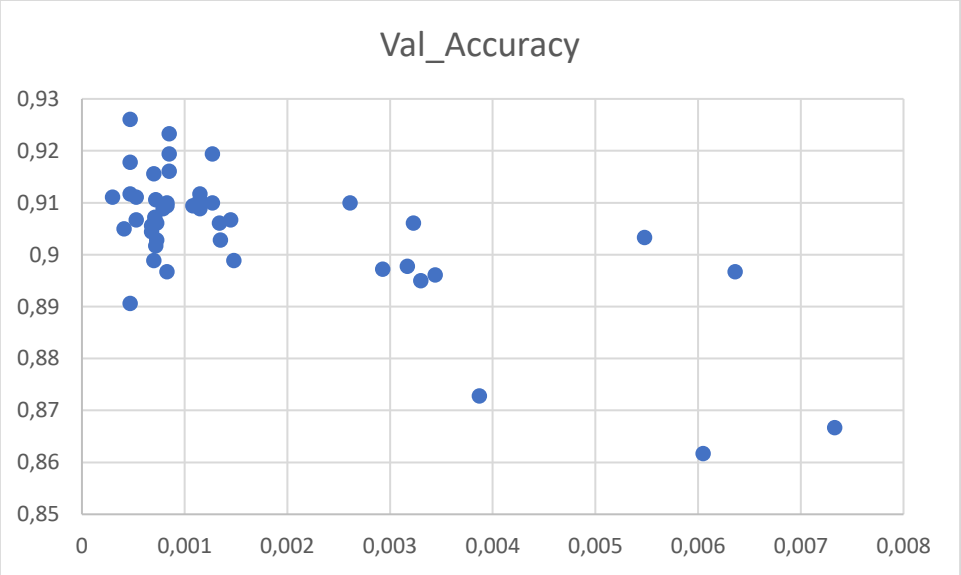
A primeira fase da análise incidiu sobre o desempenho dos algoritmos de otimização na definição da arquitetura das camadas densas. Foram realizados testes comparativos entre o Grey Wolf Optimizer (GWO) e o Random Search, mantendo-se os limites de pesquisa constantes para garantir a justiça da comparação.

Os resultados registados evidenciaram a sensibilidade do modelo, particularmente em relação à Taxa de Aprendizagem (Learning Rate). Observou-se que valores excessivamente elevados (> 0.01) impediam a convergência do modelo, resultando em precisões baixas, enquanto valores na ordem de 0.001 proporcionaram os melhores resultados.

A comparação entre os dois algoritmos revelou que:

1. Melhor Solução: O GWO conseguiu identificar uma combinação de parâmetros que maximizou a precisão de validação, demonstrando a eficácia da procura guiada por inteligência de enxame.
2. Convergência: Enquanto o Random Search explorou o espaço de forma dispersa, o GWO tendeu a concentrar a procura em regiões promissoras do espaço de híper-parâmetros.

Podemos observar isso no gráfico de dispersão abaixo:



Lucas Pagnano Golobovante – 2023140728
Diogo Valdemar Melo Estimado - 2023130197

As tabelas abaixo resumem os melhores híper-parâmetros encontrados:

GWO	423	0,16	0,00127	0,91
GWO	497	0,17	0,00047	0,9261
GWO	406	0,18	0,0007	0,8989

Random	285	0,42	0,00832	0,8806
Random	439	0,02	0,00116	0,9117
Random	51	0,02	0,00243	0,895

Análise da Sensibilidade à Redução de Dados

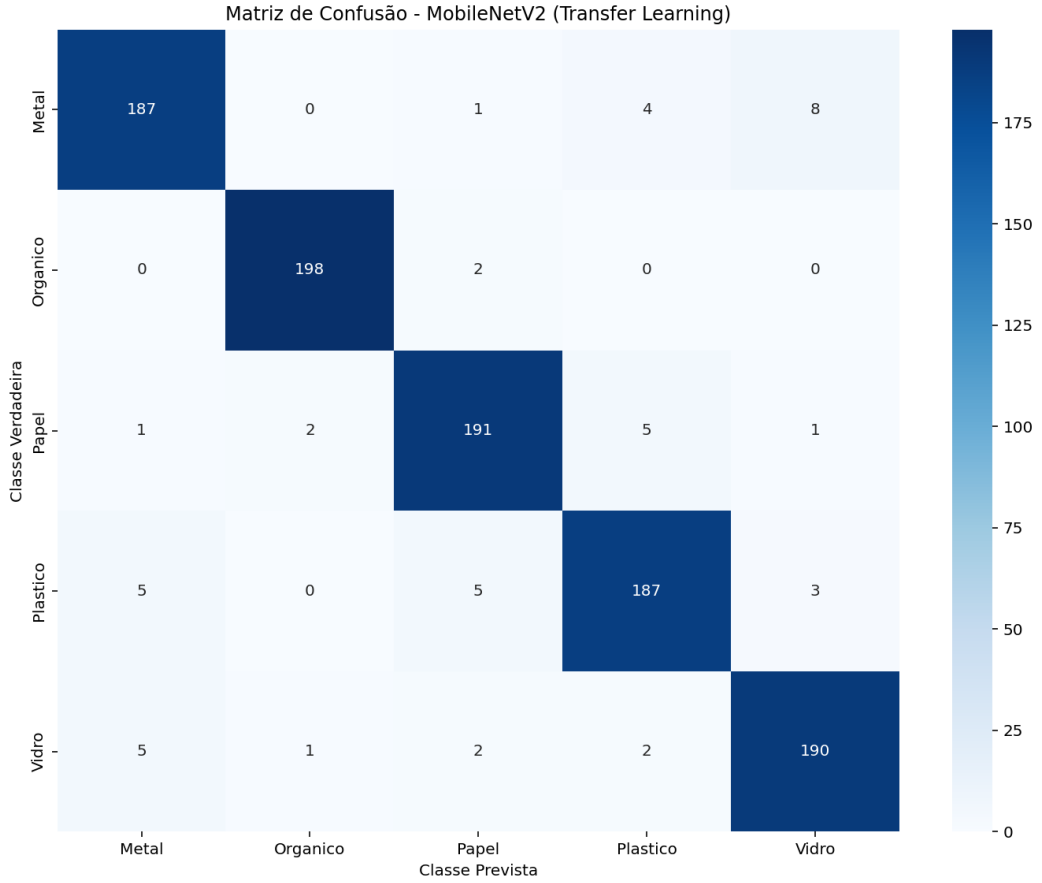
Para validar a robustez da técnica de Transfer Learning, o modelo otimizado (MobileNetV2) foi treinado com três subconjuntos de dados de dimensões distintas: 100%, 50% e 25% do conjunto de treino original. O objetivo foi avaliar o impacto do número de instâncias no desempenho final.

A análise dos resultados demonstra que a utilização de uma rede pré-treinada mitiga significativamente a perda de precisão associada à redução de dados. Mesmo com apenas 25% das imagens de treino, o modelo manteve uma performance competitiva, sofrendo apenas uma ligeira degradação em comparação com o treino completo. Isto confirma a hipótese de que a extração de características aprendida na ImageNet é altamente transferível e eficiente, dispensando a necessidade de datasets massivos para esta tarefa específica.

Dataset_Size	Num_Images	Best_Val_Accuracy	Model_File
100%	7200	0,938888907	modelo_mobilenet_100.h5
50%	3600	0,928888917	modelo_mobilenet_50.h5
25%	1800	0,903333306	modelo_mobilenet_25.h5

Avaliação no Teste

A validação final do sistema foi realizada sobre um conjunto de teste independente composto por 1000 imagens (200 por classe), garantindo uma avaliação estatisticamente significativa e balanceada. O modelo atingiu uma accuracy de 95%.



	precision	recall	f1-score	support
Metal	0.94	0.94	0.94	200
Organico	0.99	0.99	0.99	200
Papel	0.95	0.95	0.95	200
Plastico	0.94	0.94	0.94	200
Vidro	0.94	0.95	0.95	200
accuracy			0.95	1000
macro avg	0.95	0.95	0.95	1000
weighted avg	0.95	0.95	0.95	1000

Desempenho por Classe:

- A classe Orgânico obteve o melhor desempenho do sistema, com uma precisão e recall de 0.99. O modelo classificou corretamente 198 das 200 imagens. Isto sugere que as características visuais dos resíduos orgânicos (texturas irregulares, ausência de formas geométricas rígidas) são facilmente distinguíveis dos materiais recicláveis manufaturados.
- As classes Metal, Plástico e Vidro, embora com resultados muito positivos (0.94), foram as que apresentaram maior desafio.

Análise de Erros:

- Observa-se que a maior confusão ocorre na classificação de Metal, onde 8 imagens foram incorretamente classificadas como Vidro. Este erro é justificável pela semelhança visual entre os materiais: ambos apresentam superfícies reflexivas e brilhos especulares que podem confundir a rede convolucional.
- Existe também uma ligeira confusão entre Papel e Plástico (5 erros em cada direção), provável devido a embalagens de plástico que se assemelham a papel ou vice-versa.

Em suma, o F1-Score consistentemente acima de 0.94 para todas as classes comprova que o modelo não apresenta enviesamento (bias) significativo, sendo seguro para implementação na aplicação real.

NOTA: foi realizado o teste para o tamanho de dataset 50% e 25% também e foi guardado na pasta docs a matriz de confusão e o report gerado para ambos.

WebApp

Para cumprir o requisito de demonstração do sistema em ambiente real, foi desenvolvida uma aplicação web interativa utilizando a biblioteca Streamlit. Esta escolha deve-se à capacidade da framework em prototipar rapidamente interfaces de Data Science, permitindo a integração direta com o TensorFlow/Keras.

A implementação final (app.py) apresenta as seguintes características técnicas:

1. Gestão Eficiente de Recursos (Caching)

O carregamento do modelo treinado (modelo_mobilenet_100.h5) é uma operação computacionalmente pesada. Para otimizar a experiência do utilizador, utilizou-se o decorador @st.cache_resource. Esta funcionalidade garante que o modelo é carregado em memória apenas uma vez no arranque da aplicação.

2. Versatilidade de Entrada (Upload e Câmara)

A interface foi desenhada com um sistema de Abas (st.tabs), oferecendo dois modos de operação distintos:

- Modo Ficheiro: Permite o carregamento de imagens estáticas (JPG, PNG) existentes no dispositivo.
- Modo Câmara (Real-time): Utiliza o componente st.camera_input para aceder à webcam do utilizador, permitindo capturar e classificar resíduos no momento. Esta funcionalidade simula o cenário de utilização real num contentor inteligente.

3. Pipeline de Inferência Unificado

Independentemente da origem da imagem (Upload ou Câmara), os dados são encaminhados para um buffer unificado e processados pela função predict_image. Esta função replica rigorosamente o pré-processamento utilizado na fase de treino:

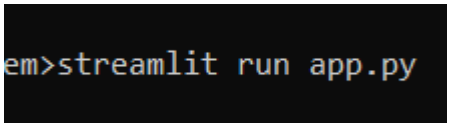
- Redimensionamento para 224x224 pixéis.
- Conversão para array e expansão de dimensões (batch size = 1).
- Normalização específica da MobileNetV2 (intervalo [-1, 1]) através da função preprocess_input.

4. Feedback Visual e Confiança

A aplicação não se limita a apresentar a classe vencedora. Implementou-se um sistema de feedback visual dinâmico baseado no nível de confiança da rede (softmax):

- Confiança > 85%: Indicador de "Alta Certeza" (Verde).
- Confiança entre 60% e 85%: Indicador "Moderada" (Amarelo).
- Confiança < 60%: Indicador de "Incerteza" (Vermelho), alertando o utilizador para possíveis erros.

Adicionalmente, é apresentado um gráfico de barras com as probabilidades de todas as 5 classes, permitindo analisar "segundas opções" do modelo em casos de dúvida.



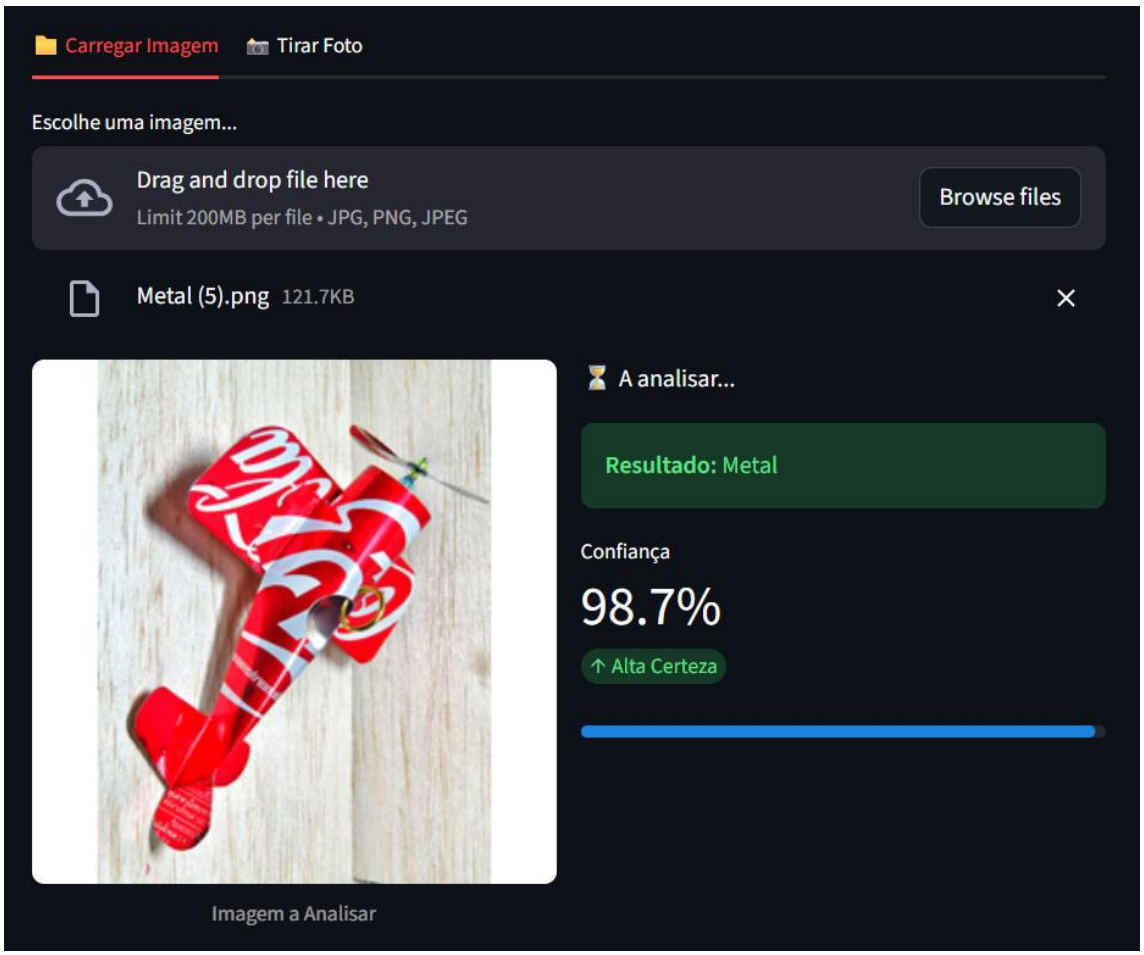
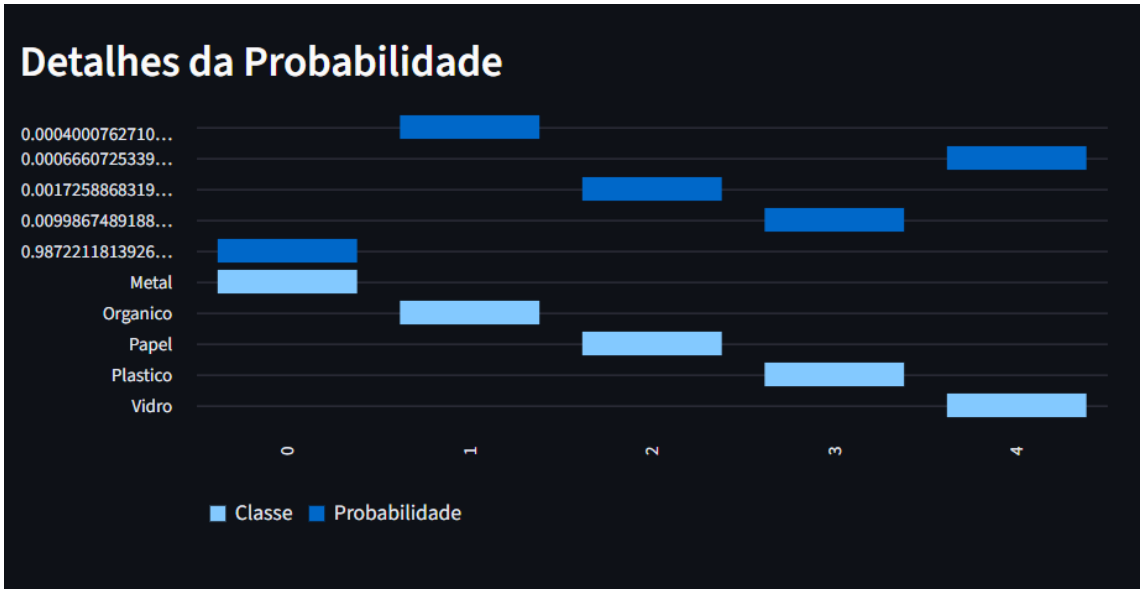
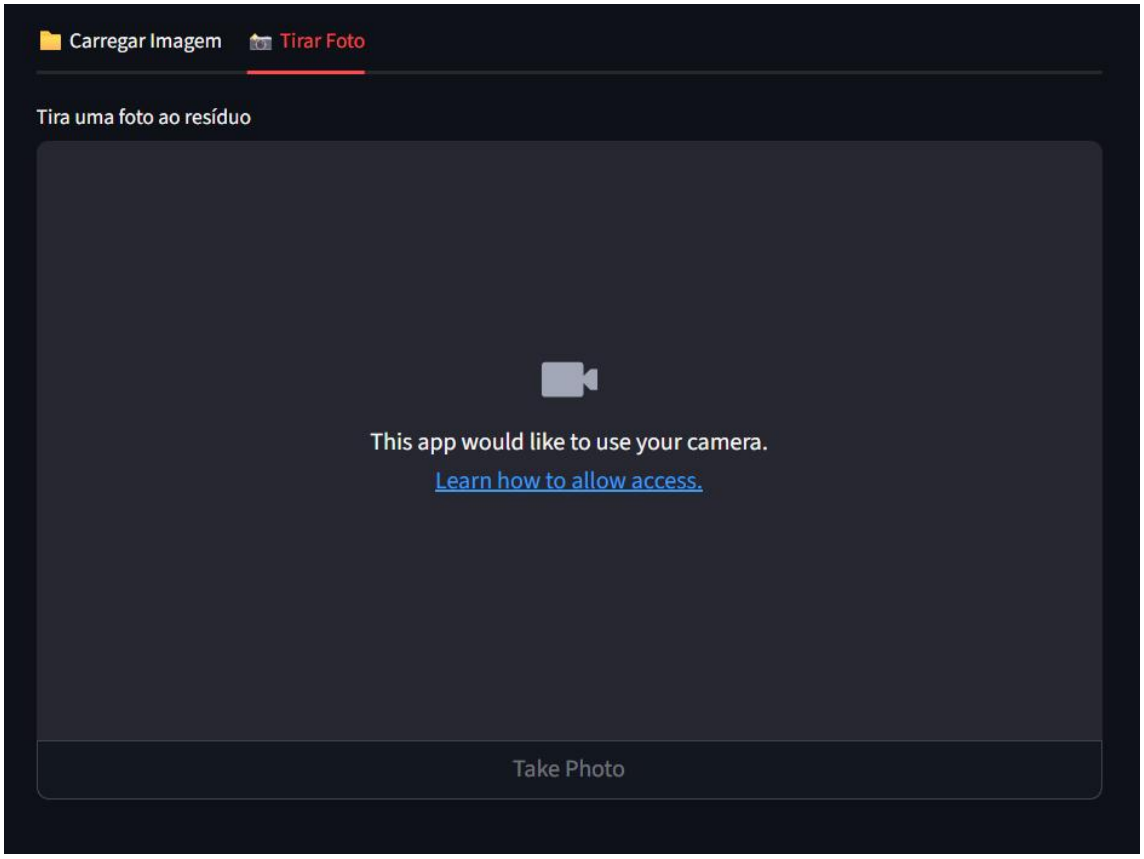


Imagem a Analisar

Lucas Pagnano Golobovante – 2023140728
Diogo Valdemar Melo Estimado - 2023130197





Lucas Pagnano Golobovante – 2023140728
Diogo Valdemar Melo Estimado - 2023130197

Conclusões

O projeto teve como objetivo o desenvolvimento de um sistema inteligente de classificação de resíduos para reciclagem, utilizando técnicas avançadas de Transfer Learning. O trabalho terminou na implementação de uma aplicação Web capaz de classificar resíduos em tempo real com elevada fiabilidade.

Síntese do trabalho

Ao longo desta Fase III, abandonou-se a abordagem de treino de redes profundas "do zero" em favor da reutilização da arquitetura MobileNetV2. Esta mudança de paradigma, aliada à otimização de híper-parâmetros via Grey Wolf Optimizer (GWO), permitiu ultrapassar as limitações de recursos computacionais e a escassez de dados que afetavam as fases anteriores.

Foi demonstrado que:

1. A utilização da função de pré-processamento adequada (preprocess_input) é crítica para o sucesso do Transfer Learning.
2. O algoritmo GWO revelou-se eficaz na afinação das camadas densas, convergindo para soluções superiores ou equivalentes à pesquisa aleatória, mas de forma mais estruturada.
3. O modelo mantém uma performance robusta mesmo quando treinado com apenas 25% dos dados, validando a capacidade de generalização das características aprendidas na ImageNet.

Resultados Alcançados

O modelo final atingiu uma precisão de 95% no conjunto de teste independente. A análise de erros revelou que as falhas são residuais e maioritariamente compreensíveis (confusão Metal/Vidro devido a reflexos), não comprometendo a utilidade prática do sistema. A aplicação Web desenvolvida cumpre o requisito de demonstrar o classificador em funcionamento num ambiente acessível ao utilizador final.

Prompt

https://gemini.google.com/share/aedb854ff180