

Classificação de Imagens com Machine Learning

Por Nicole Pessoa- Para iniciantes em ML que não querem ficar boiando 🐼

Objetivo do Projeto

Este código é como um "olho digital" 👁️ que tenta identificar se uma pessoa na foto é mulher branca ou mulher preta. Ele usa técnicas de Machine Learning (ML) e Visão Computacional para:

1. **Classificar imagens** (usando modelos pré-treinados como o ViT e o MobileNetV2).
2. **Treinar um modelo personalizado** para reconhecer características raciais em fotos.
3. **Testar imagens da internet** e dar um palpite (com porcentagem de confiança) sobre a classe racial.

👉 **Analogia:** É como ensinar uma criança a separar balas azuis e vermelhas, mas no mundo digital!

Bibliotecas Usadas (e o que elas fazem)

Biblioteca	Função	Analogia
transformers	Fornecer modelos de IA pré-treinados (como o ViT).	🏠 É como pegar um professor que já estudou muito (o modelo) e adaptá-lo para sua tarefa.
PIL (Python Imaging Library)	Manipula imagens (abrir, redimensionar, etc.).	🖼️ Tipo um Photoshop básico em código.
tensorflow keras	Cria e treina redes neurais.	🧠 O "cérebro" do ML, onde a mágica acontece.
matplotlib	Gera gráficos e mostra imagens.	📊 Um canivete suíço para visualização de dados.
os e shutil	Organiza arquivos e pastas.	📁 O organizador de arquivos do seu PC, mas programável.

Explicação Passo a Passo do Código

1 Usando Modelos Pré-Treinados (Transformers)

```
1 # Use a pipeline as a high-level helper
2 from transformers import pipeline
3 from PIL import Image
4
5
6 pipe = pipeline("image-classification", model="google/vit-base-patch32-384", device=0)
```



```
1 imagem = Image.open("jovem-mulher-negra-surpresa-com-a-boca-aberta_23-2148183287.jpg")
2 res = pipe(imagem)
3
4 res
```

- **O que acontece?**
 - `pipeline()` pega um modelo pronto (**ViT**) e classifica a imagem.
 - Ele já foi treinado para reconhecer **milhares de categorias** (não só raça).
 - `device=0` significa "use a GPU se tiver" (tipo colocar turbo no PC 🚀).
- **Resultado:**
 - Retorna uma lista de possíveis classes (ex: "mulher", "surpresa") com porcentagens de confiança.

O que é esse model?

É um **Vision Transformer (ViT)** criado pelo Google, especializado em classificar imagens. Pense nele como:

- **Um expert em reconhecer padrões visuais** 🧐 (como objetos, rostos, cenários).
- **Pré-treinado no ImageNet** (um dataset com milhões de imagens de gatos, carros, prédios, etc.).
- **"Base"** significa que é um modelo de tamanho médio (nem muito simples, nem muito complexo).

O que significam os números no nome?

- **patch32**: Divide a imagem em quadradinhos de **32x32 pixels** para análise (como montar um quebra-cabeça 🧩).
- **384**: Tamanho final da imagem que o modelo espera (**384x384 pixels**). Se você mandar uma foto maior, ele automaticamente redimensiona.

Como ele funciona?

1. **Divide a imagem** em patches (quadradinhos).
2. **Transforma cada patch** em um vetor numérico (matemática! ✨).

3. **Usa atenção global** (como se destacasse partes importantes da imagem).
4. **Classifica** com base no que aprendeu.

💡 Por que usamos ele aqui?

- **Vantagens:**
 - Ótimo para classificação genérica (não só raça, mas milhares de categorias).
 - Mais moderno que CNNs tradicionais (como o ResNet).
- **Desvantagens:**
 - Consome mais memória que modelos menores.
 - Pode ser "exagerado" para tarefas simples (como separar 2 classes).

🎯 Comparação com Outros Modelos

Modelo	Prós	Contras
ViT (este)	Alta precisão, bom para imagens complexas	Pesado, requer mais dados
ResNet	Rápido, bom para tarefas simples	Menos preciso em contextos complexos
Mobile Net	Leve (rodar até no celular)	Precisão menor

👉 **Dica:** Se estiver com poucas imagens, comece com um modelo menor como o **MobileNetV2**.

🤖 Curiosidade

O ViT foi inspirado em modelos de linguagem (como o BERT), mas adaptado para imagens! Ele "lê" patches como se fossem palavras. 📖 → 🖼️

Quer experimentar outros modelos? Troque para:

- [google/vit-base-patch16-224](#) (mais leve)
- [facebook/deit-tiny-patch16-224](#) (mini-ViT para PCs fracos)

🤖 Passo a Passo Detalhado

1 `detector(imagem)` - O "Caçador de Objetos"

- **O que faz?** Usa um modelo de detecção de objetos (no caso, o `facebook/detr-resnet-50`) para identificar tudo na foto:
 - Pessoas 🧑
 - Objetos 🚗☕
 - Animais 🐕

```
1 [{"label": 'person', 'score': 0.98, 'box': [x1, y1, x2, y2]}, ...]
```

- **label:** O que foi detectado
- **score:** Confiança do modelo (0.98 = 98% de certeza)
- **box:** Coordenadas da caixa delimitadora (onde o objeto está na imagem)

2 `[d for d in detections if d['label'] == 'person']` - Filtro de Pessoas

- **Tradução:** "Me dê a primeira caixa onde o rótulo é 'person'"
- **Por que?** Para pegar **apenas a pessoa principal** da imagem (ignorando objetos/animais)
- **Resultado em `person_bbox`:**

```
1 {'label': 'person', 'score': 0.98, 'box': [100, 150, 300, 400]}
```

(Significa que a pessoa está na área entre os pixels (100,150) e (300,400))

3 `classifier(imagem)` - O "Classificador de Atributos"

- **Modelo usado:** `google/vit-base-patch16-224` (o mesmo ViT, mas em tamanho menor)
- **O que faz?** Analisa a **imagem inteira** e tenta adivinhar:
 - Emoções 😲
 - Ações (ex: "surpresa")
 - Características físicas
- **Exemplo de saída:**

```
1 [{"label": 'surprised', 'score': 0.92}, {"label": 'woman', 'score': 0.89}]
```

Diferença Entre as Duas Abordagens

Função	Objetivo	Modelo Usado	Retorno
<code>detector()</code>	Achar onde estão objetos/pessoas	DETR (ResNet-50)	Caixas delimitadoras + rótulos
<code>classifier()</code>	Entender o que a imagem representa	ViT (Google)	Lista de características + confiança

Problemas Comuns/Erros

1. Se não achar pessoas:
 - `person_bbox = [d for d in detections if d['label'] == 'person']` → `IndexError`
 - **Solução:** Verifique se `detections` tem itens com `'person'` antes de acessar.
 2. Classificação errada:
 - O classificador pode confundir "surpresa" com "medo" ou "mulher" com "homem" se a foto for ambígua.
 3. Falsa precisão:
 - Modelos pré-treinados podem ter **viés racial/gênero** (ex.: classificar mais errado para pele escura).
-

Passo a Passo Detalhado

1 `print("\n🔍 Resultados da Classificação:")`

- `\n` cria uma linha em branco antes do título
- 🔍 é um emoji de lupa para deixar visual
- Mostra o cabeçalho "Resultados da Classificação"

2 `print("=" * 40)`

- Cria uma linha divisória com 40 sinais de =
- Exemplo: =====

3 O Loop `for item in result:`

- Itera por cada classificação encontrada no resultado
- Para o exemplo acima, vai rodar 3 vezes (uma para 'surprised', 'young' e 'woman')

4 `print(f"🏷️ {item['label'].upper():<30} | 📊
{item['score']*100:.2f}%")`

- 🏷️ - Emoji de etiqueta
 - `item['label'].upper()` - Pega o rótulo e coloca em MAIÚSCULAS (ex: 'SURPRISED')
 - `:<30` - Formata para ocupar 30 caracteres (alinhado à esquerda)
 - `|` - Uma barra vertical separadora
 - 📊 - Emoji de gráfico
 - `item['score']*100:.2f` - Converte o score (0.92) para porcentagem (92.00%) com 2 casas decimais
-

🎨 Por que essa formatação?

1. Visual Clean - Facilita a leitura dos resultados
2. Alinhamento - Os `:<30` mantém as colunas alinhadas mesmo com textos de tamanhos diferentes
3. Emojis - Tornam a saída mais amigável e intuitiva
4. Divisórias - As linhas de `=` delimitam claramente o bloco de resultados

🤔 Perguntas Frequentes

P: Por que multiplicar o score por 100?

R: Porque os scores vêm em decimal (0-1) e queremos mostrar como porcentagem (0-100%)

P: O que significa `:30` no formato?

R: É o número mínimo de caracteres que a string deve ocupar, alinhado à esquerda

P: Posso usar isso para outros modelos?

R: Sim! Funciona com qualquer classificador que retorne uma lista de dicionários com 'label' e 'score'

```
1 pip install torch torchvision matplotlib
```

este comando instala três bibliotecas essenciais para trabalhar com Machine Learning e visualização de dados em Python. Vamos destrinchar cada uma:




1 torch - O "Superpoder" do Deep Learning

- **O que é?** A biblioteca principal do **PyTorch**, um dos frameworks mais populares para Deep Learning
 - **Para que serve?**
 - Cria e treina redes neurais complexas 🧠
 - Faz cálculos tensoriais (matrizes multidimensionais) de forma acelerada (GPU/CPU) ⚡
 - Usada por pesquisadores e empresas (como Meta/Facebook)
 - **Analogia:** É como o motor de um carro de F1 da IA 🏎️🧠
-

2 torchvision - O "Kit de Visão Computacional"

- **O que é?** Uma biblioteca irmã do PyTorch especializada em:
 - Processamento de imagens 📷
 - Modelos pré-treinados (ResNet, ViT, etc.) 🏗️
 - Datasets populares (MNIST, CIFAR-10) 📚

3 matplotlib - A "Caneta Mágica" da Visualização

- **O que é?** A biblioteca mais usada para criar gráficos em Python
- **Para que serve no ML?**
 - Plotar curvas de aprendizado 
 - Visualizar imagens e resultados 
 - Criar heatmaps de atenção 

Por que instalar esses três juntos?

1. **torch** faz o trabalho pesado de ML
2. **torchvision** lida com a parte visual
3. **matplotlib** mostra os resultados

Juntas, formam o "trio de ouro" para projetos de visão computacional!

```
import os

path = "/content/drive/MyDrive/Projeto ML/Dataset-img"
print(os.listdir(path))
```

O que esse código faz?

1. **Importa** o módulo **os** (que interage com o sistema operacional)
2. **Define** o caminho (path) de uma pasta no Google Drive
3. **Lista** todos os arquivos/diretórios dentro dessa pasta
4. **Imprime** essa lista

Exemplo Prático

Se sua pasta contiver:

```
Dataset-img/
├── fotos-grupo/
├── mulheres-brancas/
└── mulheres-negras/
```

```
import shutil
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
```

Vamos desvendar cada parte dessas importações essenciais para criar um modelo de visão computacional:

1 `import shutil` - O "Ajudante de Arquivos"

- **Função:** Operações de alto nível com arquivos e pastas
- **Uso no ML:**

```
shutil.copy(src, dst)    # Copia arquivos entre pastas
shutil.rmtree(path)      # Remove árvore de diretórios
```

Por que importar? Para organizar datasets antes do treino

2 `import tensorflow as tf` - O "Cérebro" do Deep Learning

- **O que é?** Framework completo para ML
 - **Principais submodules:**
 - `tf.keras`: API de alto nível para redes neurais
 - `tf.data`: Pipelines eficientes de dados
 - `tf.image`: Processamento de imagens
-

3 `ImageDataGenerator` - O "Fábrica de Imagens"

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Para que serve? Gera batches de imagens com:

- **Data augmentation** (aumento artificial do dataset)
- **Pré-processamento automático**

Exemplo clássico:

```
datagen = ImageDataGenerator(rotation_range=20, zoom_range=0.2)
```

4 Sequential - O "Construtor de Modelos"

```
from tensorflow.keras.models import Sequential
```

Analogia: Como montar uma casa peça por peça 🏠

Uso típico:

```
1 model = Sequential([
2     Conv2D(32, (3,3), activation='relu'),
3     MaxPooling2D(),
4     Flatten(),
5     Dense(10, activation='softmax')
6 ])
```

5 Camadas Essenciais - Os "Tijolos" da Rede Neural

```
1 from tensorflow.keras.layers import (
2     Conv2D,      # Extrai features das imagens
3     MaxPooling2D, # Reduz dimensionalidade
4     Flatten,     # "Achata" os dados para Dense
5     Dense        # Camadas totalmente conectadas
6 )
```

Detalhes das Camadas:

Camada	Função	Parâmetros Chave
<code>Conv2D</code>	Detecta padrões	<code>filters</code> , <code>kernel_size</code>
<code>MaxPooling2D</code>	Reduz tamanho	<code>pool_size</code>
<code>Flatten</code>	Prepara para Dense	-
<code>Dense</code>	Classificação	<code>units</code> , <code>activation</code>



Pipeline Completo de Trabalho

1. `shutil` organiza os dados
 2. `ImageDataGenerator` prepara as imagens
 3. `Sequential` constrói o modelo
 4. Camadas (`Conv2D`, etc.) definem a arquitetura
-

2 Organizando as Imagens para Treino



Explicação da Função `organize_images()`

Esta função é como um **organizador de arquivos inteligente** para projetos de Machine Learning com imagens. Vamos entender peça por peça:



Objetivo Principal

Organizar imagens em uma estrutura padronizada que o Keras/TensorFlow consegue entender automaticamente, no formato:

pasta_principal/

└─ classe_1/ # Ex: 'mulheres-brancas'

└─┬─ img1.jpg

└─┬─ img2.jpg

Anatomia da Função

1. Parâmetros

```
# 1. Primeiro, vamos organizar suas imagens
def organize_images(source_dir, target_dir, class_name):
```

- `source_dir`: Pasta de origem das imagens (onde estão bagunçadas)
- `target_dir`: Pasta destino principal (onde vai ficar organizado)
- `class_name`: Nome da classe/subpasta (ex: 'brancas')

2. Criando a Pasta Destino

```
os.makedirs(os.path.join(target_dir, class_name), exist_ok=True) #Cria a pasta de destino (se não existir):
```

- `os.path.join()`: Monta caminhos de forma correta para qualquer sistema operacional
- `exist_ok=True`: Não dá erro se a pasta já existir (útil para reexecução)

3. Loop pelos Arquivos

```
for img_file in os.listdir(source_dir): #Percorre todos os arquivos na pasta original:
```

- `os.listdir()`: Lista tudo dentro da pasta origem
- Itera em cada item (arquivo ou subpasta)

4. Filtro de Arquivos de Imagem

```
if img_file.lower().endswith(('.png', '.jpg', '.jpeg', '.webp', '.avif')): #Verifica se é imagem
```

- Verifica a extensão do arquivo (case insensitive por causa do `.lower()`)
- Suporta formatos comuns: PNG, JPG, JPEG, WEBP, AVIF

5. Cópia Segura

```
src = os.path.join(source_dir, img_file) # Caminho original
dst = os.path.join(target_dir, class_name, img_file) # Caminho novo
shutil.copy(src, dst) # Cópia (não move)
```

- **src**: Caminho completo da imagem original
- **dst**: Caminho completo de destino
- **shutil.copy()**: Cópia sem alterar o original (mais seguro que mover)



Organizando as Imagens para Treino

```
organize_images(source_dir_brancas, os.path.join(target_dir, 'treinamento'), 'brancas')
organize_images(source_dir_pretas, os.path.join(target_dir, 'treinamento'), 'pretas')
```

O que está acontecendo?

1. **Duas chamadas idênticas** da função `organize_images()` para classes diferentes:
 - Uma para imagens de mulheres brancas ('brancas')
 - Outra para mulheres pretas ('pretas')
2. **Estrutura resultante:**

```
dataset_organizado/
├── treinamento/
│   ├── brancas/
│   │   ├── foto1.jpg
│   │   └── foto2.jpg
│   └── pretas/
│       └── foto1.jpg
```

1. └── foto2.jpg
2. **Por que `os.path.join`?**
 - Garante compatibilidade entre Windows/Linux
 - Evita problemas com barras (/ vs \)



Configuração do Treinamento

```
# 2. Agora o treinamento
IMG_SIZE = (224, 224)
BATCH_SIZE = 8 # Reduzi porque tem poucas imagens
EPOCHS = 15 # Uma passagem completa por todo o conjunto de treinamento.
```

1. **IMG_SIZE = (224, 224)**

- **O que faz?** Redimensiona todas as imagens para 224x224 pixels
- **Por que 224?** Tamanho padrão para modelos como MobileNet/VGG16
- **Impacto:**
 - Menor = processamento mais rápido, mas menos detalhes
 - Maior = mais detalhes, mas consome mais memória

2. **BATCH_SIZE = 8**

- **Significado:** Número de imagens processadas de uma vez
- **Regra prática:**
 - GPUs pequenas: 8-16
 - GPUs potentes: 32-256
- **No código:** Foi reduzido porque há poucas imagens disponíveis

3. **EPOCHS = 15**

- **Definição:** Número de vezes que o modelo verá todo o dataset
- **Analogia:** Como repetir exercícios para aprender:
 - Poucas épocas = subaprendizado
 - Muitas épocas = overfitting (decoreba)
- **Dica:** Comece com 10-20 e ajuste conforme a evolução da acurácia

Por Que Esses Valores?

Parâmetro	Valor Típico	Nosso Código	Motivo
Tamanho	224-512	224	Balanceia desempenho/acuracia
Batch	16-256	8	Dataset pequeno
Épocas	10-100	15	Evitar overfitting

Explicação do Data Augmentation e Geradores de Dados


Vamos entender essa parte essencial do pipeline de treinamento:

Data Augmentation (Aumento de Dados)

O `ImageDataGenerator` é como um **personal trainer de imagens** - ele cria variações das suas fotos para melhorar o treinamento:

Configurações Principais

```
# Data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255, # Normaliza os valores dos pixels para [0,1]
    validation_split=0.2, # Reserva 20% dos dados para validação
    rotation_range=20, # Rotaciona imagens aleatoriamente até 20 graus
    width_shift_range=0.2, # Desloca imagem horizontalmente (20% da largura)
    height_shift_range=0.2, # Desloca imagem verticalmente (20% da altura)
    horizontal_flip=True # Inverte imagem horizontalmente aleatoriamente
)
```

Parâmetro	Efeito	Visualização	Por que Usar?
<code>rescale=1./255</code>	Normaliza pixels para [0,1]	[0,255] → [0,1]	Ajuda na convergência do modelo
<code>validation_split=0.2</code>	Separa 20% para validação	Treino (80%) vs Validação (20%)	Monitora overfitting
<code>rotation_range=20</code>	Gira imagem aleatoriamente até 20°		Modelo aprende ângulos diferentes
<code>width_shift_range=0.2</code>	Move horizontalmente (até 20% da largura)	↔	Tolerância a posicionamento
<code>height_shift_range=0.2</code>	Move verticalmente (até 20% da altura)	↑↓	Tolerância a enquadramento
<code>horizontal_flip=True</code>	Espelha a imagem horizontalmente	↔	Dobra virtualmente o dataset

Geradores de Dados

```
train_generator = train_datagen.flow_from_directory(  
    os.path.join(target_dir, 'treinamento'), # Pasta com imagens  
    target_size=IMG_SIZE, # Redimensiona imagens  
    batch_size=BATCH_SIZE, # Número de imagens por lote  
    class_mode='binary', # Classificação binária  
    subset='training' # Usa parte de treino (80%)  
)
```

Parâmetros Chave

1. **target_size=IMG_SIZE**
 - Redimensiona todas as imagens para (224, 224)
 - Fundamental para redes neurais que exigem tamanho fixo
2. **batch_size=BATCH_SIZE**
 - Define quantas imagens são processadas por vez (8 no código)
 - GPUs pequenas precisam de batches menores
3. **class_mode='binary'**
 - Para problemas com 2 classes (brancas/pretas)
 - Usaria 'categorical' para 3+ classes
4. **subset='training'**
 - Especifica que queremos os dados de TREINO (80%)
 - Usaríamos 'validation' para pegar os 20% de validação

Explicação Detalhada da Arquitetura e Treinamento do Modelo CNN

Vamos desmontar essa rede neural convolucional (CNN) peça por peça:

Arquitetura do Modelo (Sequential API)

1. Primeira Camada Convolucional

```
# Primeira camada convolucional  
Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3)),
```

- **32 filtros:** Cria 32 mapas de características diferentes
- **Kernel 3x3:** Janela que desliza pela imagem detectando padrões
- **ReLU (Rectified Linear Unit):** $\max(0, x)$ - Introduce não-linearidade

- **input_shape:** (altura, largura, canais) para imagens RGB

2. Camada de Max Pooling

```
MaxPooling2D(2, 2), # Redução de dimensionalidade
```

- **Redução 2x2:** Mantém apenas o valor máximo em cada janela 2x2
- **Objetivo:**
 - Reduz dimensionalidade
 - Mantém características importantes
 - Controla overfitting

3. Segunda Camada Convolucional

```
Conv2D(64, (3, 3), activation='relu'), # Segunda camada convolucional
```

- **64 filtros:** Aumenta capacidade de aprendizado
- **Detecta padrões mais complexos:** Combina features das camadas anteriores

4. Achatamento (Flatten)

Flatten()

- Transforma o tensor 3D (altura, largura, canais) → vetor 1D
- Exemplo: (28, 28, 64) → 50176 valores

5. Camadas Densas (Fully Connected)

```
Dense(128, activation='relu'), # Camada densa (fully connected)
Dense(1, activation='sigmoid') # Camada de saída (classificação binária)
```

- **128 neurônios:** Camada oculta para decisão complexa
- **Saída binária:** 1 neurônio com sigmoide (probabilidade 0-1)

Configuração do Treinamento

Otimizador Adam

```
optimizer='adam', # Otimizador eficiente
```

- **Adaptive Moment Estimation**
- Combina ideias de Momentum + RMSProp
- Taxa de aprendizado adaptativa por parâmetro

Função de Perda

```
loss='binary_crossentropy', # Função de perda para classificação binária
```

- Ideal para classificação binária
- Calcula diferença entre previsões e labels reais
- Fórmula: $-(y \cdot \log(p) + (1-y) \cdot \log(1-p))$

Métrica

```
metrics=['accuracy'] # Métrica a ser monitorada
```

- Acompanha a % de acertos durante treino/validação
- Outras opções: precision, recall, F1-score

Processo de Treinamento

Método fit()

```
# Treinamento
history = model.fit(
    train_generator, # Dados de treino
    validation_data=validation_generator, # Dados de validação
    epochs=EPOCHS # Número de épocas
)
```

- **Generators:** Fornecem batches de dados com augmentation
- **Épocas:** Passagens completas pelo dataset
- **Histórico:** Armazena métricas para visualização

Explicação do Modelo MobileNetV2 com Transfer Learning

Vamos explorar essa técnica poderosa de visão computacional:

O que é Transfer Learning?

Técnica que aproveita um modelo pré-treinado (no caso, no ImageNet) como ponto de partida para seu problema específico. É como pegar um professor universitário e adaptá-lo para ensinar ensino médio!

Montando o Modelo Passo a Passo

1. Importando a Base MobileNetV2

```
# Carrega o modelo base MobileNetV2 com pesos pré-treinados no ImageNet
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```
base_model = MobileNetV2(
    weights='imagenet', # Carrega pesos pré-treinados
    include_top=False, # Remove a camada de classificação original
    input_shape=(224, 224, 3) # Define o formato de entrada
)
```

Por que MobileNetV2?

- Leve e eficiente (ótimo para dispositivos móveis)
- Pré-treinado em 1.4 milhão de imagens (ImageNet)
- Arquitetura otimizada para velocidade

2. Congelando os Pesos

```
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False # Os pesos não serão atualizados durante o treino
```

Motivação:

- Mantém os padrões genéricos aprendidos no ImageNet
- Evita sobrescrever conhecimento útil
- Acelera o treinamento (só treina as novas camadas)

3. Construindo o Novo Modelo

```
# Cria um novo modelo sequencial
model = Sequential([
    base_model, # Extrai features das imagens
    Flatten(), # Transforma os features em vetor
    Dense(128, activation='relu'), # Adiciona capacidade de aprendizado
    Dense(1, activation='sigmoid') # Saída entre 0 e 1 (probabilidade)
])
```

Anatomia do Modelo:

1. **Extrator de Features:** MobileNetV2 congelado
2. **Adaptador:** Flatten + Dense(128) para aprender combinações específicas
3. **Classificador:** Única saída com sigmoide (0 ou 1)

⚡ Vantagens Dessa Abordagem

1. **Eficiência:** Treina muito mais rápido que do zero
2. **Performance:** Aproveita features robustas do ImageNet
3. **Dataset Pequeno:** Funciona bem com poucas imagens (100-1000 por classe)



Comparação com o Modelo Anterior

Característica	CNN Manual	MobileNetV2
Velocidade	Mais lento	Mais rápido
Precisão	Menor	Maior (em geral)
Requer Dados	Mais imagens	Funciona com menos
Flexibilidade	Total	Limitada pela arquitetura base



Função para Processar Imagens da Internet

Vamos explicar esse código que faz magia com imagens da web:



Como Funciona Passo a Passo

1. Carregando o Modelo Treinado

```
# Carregar o modelo treinado
model = load_model('modelo_classificacao_racial.h5')
```

- **O que faz?** Carrega seu modelo de machine learning já treinado
- **Formato .h5** - É como um "container" que guarda toda a inteligência do seu modelo
- **Importante:** O modelo deve ter a mesma estrutura de quando foi salvo

2. A Função Mágica

```
# Função para processar imagem da internet
def test_image_from_url(img_url):
```

Essa função é tipo um **chef de cozinha** que:

1. Pega a imagem da internet 🌐
2. Prepara ela direitinho 🍴
3. Mostra pra você 👁️

A. Baixando a Imagem

```
# Baixar a imagem
response = requests.get(img_url) # Timeout de 10 segundos
img = Image.open(BytesIO(response.content)) # Verifica erros HTTP
```

- `requests.get` - Pega a imagem da web (como salvar uma foto do Instagram)
- `BytesIO` - Transforma os dados baixados num formato que o Python entende

B. Convertendo para RGB

```
# Converte para RGB caso seja PNG, RGBA, etc.
if img.mode != 'RGB':
    | | img = img.convert('RGB')
```

- **Por quê?** Algumas imagens vem com:
 - Transparência (RGBA)
 - Preto e branco
- Seu modelo só entende RGB (vermelho, verde, azul)

C. Mostrando a Imagem

```
# Mostrar a imagem original
plt.figure(figsize=(5, 5))
plt.imshow(img)
plt.axis('off')
plt.title('Imagem Original')
plt.show()
```

- `figsize=(5,5)` - Tamanho da janela (5 polegadas)
- `axis('off')` - Tira aqueles eixos chatos
- `title()` - Coloca um nome em cima da imagem



Cuidados Importantes

1. **URLs maliciosas** - Pode ser vírus! Sempre valide o link
 2. **Tamanho da imagem** - Imagens muito grandes podem travar
 3. **Direitos autorais** - Não use imagens que não pode
-



Processamento e Predição de Imagens - Explicação Detalhada

Vamos desvendar cada etapa desse código essencial para classificação de imagens:



Pré-Processamento da Imagem

1. Redimensionamento

```
# Pré-processamento  
img = img.resize((224, 224)) # Redimensiona para o tamanho esperado pelo modelo
```

- **Por que?** Seu modelo foi treinado com imagens 224x224 pixels
- **Problema evitado:** Erros de dimensão incompatível
- **Dica:** Mantenha a proporção original com `thumbnail()` se necessário

2. Conversão para Array Numérico

```
img_array = image.img_to_array(img) # Converte para array numpy
```

- Transforma a imagem em uma matriz NumPy (altura × largura × canais)
- Formato resultante: (224, 224, 3) para imagens RGB

3. Adição de Dimensão do Batch

```
img_array = np.expand_dims(img_array, axis=0) # Adiciona dimensão do batch
```

- Transforma para (1, 224, 224, 3)
- **Motivo:** Modelos Keras esperam um batch (lote) de imagens, mesmo que seja só uma

4. Normalização

```
img_array /= 255.0 # Normaliza pixels para [0,1]
```

- Converte valores de pixel de 0-255 para 0-1
- **Benefício:** Ajuda o modelo a convergir mais rápido

Fazendo a Predição

1. Chamada do Modelo

```
# Fazer a predição  
prediction = model.predict(img_array) # Suprime logs  
prob_branca = 1 - prediction[0][0] # Probabilidade de
```

- Retorna um array com as probabilidades
- Formato: `[[probabilidade]]` (ex: `[[0.73]]`)

2. Interpretação das Probabilidades

```
prob_branca = 1 - prediction[0][0] # Probabilidade da classe 'preta'  
prob_preta = prediction[0][0] # Probabilidade complementar
```

- Modelo binário: Probabilidade sempre entre 0 e 1
- Se saída $\approx 0 \rightarrow$ Classe A ("Preta")
- Se saída $\approx 1 \rightarrow$ Classe B ("Branca")

3. Determinação da Classe

```
# Determinar a classe  
if prob_branca > prob_preta:  
    classe = "Branca"  
    confidence = prob_branca  
else:  
    classe = "Preta"  
    confidence = prob_preta
```

- **Threshold:** 0.5 (50%) é o ponto de corte padrão
 - **confidence:** Mostra quão seguro o modelo está
-

```
# Mostrar resultados
print("\nResultado da Classificação:")
print(f"Classe Predita: {classe}")
print(f"Confiança: {confidence:.2%}")
print(f"Probabilidade Branca: {prob_branca:.2%}")
print(f"Probabilidade Preta: {prob_preta:.2%}")
```

Dica do dia: O `\n` serve para pular uma linha antes da mensagem, organizando melhor a saída.

`print(f"Classe Predita: {classe}")` - **classe** (ou categoria) que o modelo escolheu.

Ex: "Classe Predita: branca"

```
print(f"Confiança: {confidence:.2%}")
```

Essa linha exibe a **confiança** do modelo na predição.

`.2%` formata o número como porcentagem com 2 casas decimais.

Ex: 0.87 →

```
print(f"Probabilidade Branca: {prob_branca:.2%}")
```

```
print(f"Probabilidade Preta: {prob_preta:.2%}")
```

Essas linhas mostram a **probabilidade individual** atribuída a cada uma das classes.

Isso ajuda a entender se a decisão foi apertada (ex: 51% x 49%) ou clara (ex: 90% x 10%).

```
    return {
        'class': classe,
        'confidence': float(confidence),
        'probabilities': {
            'branca': float(prob_branca),
            'preta': float(prob_preta)
        }
    }
```

Esta parte devolve (retorna) os dados como um **dicionário**.

Isso é útil para quem quiser usar esse resultado em outras partes do código, como para gravar num banco de dados ou exibir em uma interface web.

Por que isso é útil?

1. **Dicionário Python** - Facilita acesso programático aos resultados
2. **Valores convertidos** - De `numpy.float32` para `float` nativo
3. **Estrutura clara** - Pode ser convertido para JSON fácil

```
except Exception as e:
    print(f"Erro ao processar a imagem: {e}")
    return None
```

Se alguma coisa der errado durante o processamento da imagem (por exemplo, se o arquivo estiver corrompido), o programa **não trava**: ele imprime o erro e retorna `None`.

🚒 Isso é como um plano de emergência para evitar que o sistema quebre.

Boas Práticas:

1. **Captura qualquer exceção** - Desde timeout até erros no modelo
2. **Log claro** - Mostra o erro específico
3. **Retorno None** - Permite verificação fácil no código chamador



Loop de Teste Automatizado - Explicação Detalhada

Vamos entender como esse código testa automaticamente várias imagens da internet:


```
# Testar cada URL
for url in test_urls:
    print(f"\nTestando imagem: {url}")
    result = test_image_from_url(url)
    if result:
        print(f"Resultado: {result['class']} (Confiança: {result['confidence']:.2%})")
```

Linha por Linha

for url in test_urls:

-  *Tradução:* “Para cada URL na lista de `test_urls`, faça o seguinte...”
-  Aqui estamos **percorrendo uma lista de links de imagens** que queremos testar no modelo.

print(f"\nTestando imagem: {url}")

-  Mostra qual imagem o modelo está testando no momento.
- O `\n` é só para dar uma quebrinha de linha, organizando melhor a visualização.

```
print(f"\nTestando imagem: {url}")
```

- 🔍 Mostra qual imagem o modelo está testando no momento.
- O `\n` é só para dar uma quebrinha de linha, organizando melhor a visualização.

🎯 O que essa parte ensina?

- Como **automatizar testes** com múltiplas entradas (várias imagens).
- Como **usar funções em loops** para testar seu modelo com diferentes dados.
- Como apresentar os resultados de maneira **clara, legível e útil para tomada de decisão**.

💡 Por Que Isso é Útil?

1. **Teste Rápido** - Verifica várias imagens de uma vez
2. **Comparação Fácil** - Veja como o modelo se comporta com diferentes fotos
3. **Debug Visual** - Identifica rapidamente URLs problemáticas

🚨 Cuidados Importantes

1. **Limite o Número de URLs** - Muitas requisições podem bloquear seu IP
 2. **Trate Erros de Conexão** - Alguns sites bloqueiam scrapers
 3. **Respeite Direitos Autorais** - Não use imagens protegida
-
-

🚀 Como Aprender Machine Learning?

Para você entender de ML, você tem que percorrer antes com:

- 1 **Lógica de Programação (Python)**
- 2 **Matemática Básica (Estatística, Álgebra)**
- 3 **Biblioteca de Python (Principalmente com pandas)**
- 4 **Visualização de Dados**

⚠ Limitações e Cuidados

- Esse modelo **não é perfeito** e pode cometer erros (viés racial existe em ML!).
- **Nunca use em produção** sem testar muito antes.
- Machine Learning é uma ferramenta poderosa, mas **requer responsabilidade**.
- Lembre-se que estamos usando pouco conjunto de dados de foto para treinar então ele vai errar na identificação
- Isso é para você entender um pouco sobre processamento de imagem

🎉 Conclusão

Agora você já sabe como:

- ✓ Usar modelos prontos do **transformers**.
- ✓ Treinar um classificador de imagens do zero.
- ✓ Testar com fotos da internet.

👉 **Dica final:** Machine Learning é como cozinhar você erra, ajusta os temperos (hiperparâmetros) e tenta de novo! 🔍