🩺 Classificação de Câncer de Pele utilizando CNN

Este notebook realiza o treinamento, avaliação e visualização com Grad-CAM de um modelo baseado em **MobileNetV2** para classificação de imagens dermatológicas.

Todas as seções estão nomeadas em português para facilitar a compreensão do fluxo de trabalho.

📦 Importação das Bibliotecas

In [1]:
```
# Instalar dependências (execute apenas se necessário)
!pip install -q pandas numpy matplotlib seaborn scikit-learn tensorflow shap ope
```

In [2]:
```
import os
from pathlib import Path
import glob
import numpy as np
import pandas as pd
from PIL import Image
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, i
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam
import cv2
import warnings
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

warnings.filterwarnings("ignore", category=UserWarning, module="keras.src.traine

# Garantir comportamento determinístico
np.random.seed(42)

print('Imports carregados')
```

```
Imports carregados
```

📂 Carregamento e Preparação dos Dados

In [3]:
```
# Detectar automaticamente o dataset e realizar download

def find_latest_kagglehub_dataset(base_dir_name='farjanakabirsamanta/skin-cancer
    home = Path.home()
    cache_base = home / '.cache' / 'kagglehub' / 'datasets'
    if cache_base.exists():
```

```
        candidates = list(cache_base.glob(f'**/{base_dir_name}'))
        if not candidates:
            candidates = list(cache_base.glob('**/skin-cancer-dataset'))
        if candidates:
            # ordenar por profundidade e escolher o primeiro mais provável
            candidates.sort(key=lambda p: (str(p).count(os.sep), p.name), revers
            return Path(candidates[0])
    # fallback: procurar no diretório atual
    cwd = Path.cwd()
    csvs = list(cwd.glob('**/HAM10000_metadata.csv'))
    if csvs:
        return csvs[0].parent
    raise FileNotFoundError('Dataset HAM10000 não encontrado. Coloque o CSV e im

dataset_base = find_latest_kagglehub_dataset()
```

In [4]:

```
# Carregar metadados e resolver caminhos das imagens
metadata_path = list(dataset_base.glob('**/HAM10000_metadata.csv'))[0]
metadata = pd.read_csv(metadata_path)

# Mapear diagnósticos
mapping = {
    'akiec': 'Queratose actínica',
    'bcc': 'Carcinoma basocelular',
    'bkl': 'Lesão benigna tipo queratose', # Ou: Queratose seborreica (dependena
    'df': 'Dermatofibroma',
    'mel': 'Melanoma',
    'nv': 'Nevo melanocítico', # Popularmente 'pinta'
    'vasc': 'Lesão vascular'
}
metadata['diagnosis'] = metadata['dx'].map(mapping)

## Renomear Colunas
# Mapeamento do nome da coluna original (inglês/código) para o nome desejado (Pt
column_rename_map = {
    'lesion_id': 'id_lesao',
    'image_id': 'image_id',
    'dx': 'codigo_diagnostico', # O código de diagnóstico original (ex: nv, mel)
    'dx_type': 'tipo_diagnostico',
    'age': 'idade',
    'sex': 'sexo',
    'localization': 'localizacao',
    'diagnosis': 'diagnostico', # O diagnóstico mapeado por extenso
    'image_path': 'image_path'
}

# Aplicar o renomeio
metadata = metadata.rename(columns=column_rename_map)

# localizar pasta de imagens: procurar pastas com muitas imagens
image_folders = [p for p in dataset_base.glob('**/*') if p.is_dir()]
best = None
best_count = 0
for p in image_folders:
    count = len(list(p.glob('*.jpg')))
    if count > best_count:
        best = p
        best_count = count
if best is None or best_count == 0:
    raise FileNotFoundError('Pasta de imagens não encontrada no dataset base: '
```

```
images_root = best

# construir caminho completo

def resolve_image_path(image_id, images_root):
    for ext in ('.jpg', '.jpeg', '.png'):
        candidate = images_root / f"{image_id}{ext}"
        if candidate.exists():
            return str(candidate)
    # procurar recursivamente
    found = list(images_root.rglob(f"{image_id}.*"))
    return str(found[0]) if found else None

metadata['image_path'] = metadata['image_id'].apply(lambda x: resolve_image_path
metadata = metadata[metadata['image_path'].notnull()].reset_index(drop=True)

# Mapeamento dos valores de sexo
sexo_mapping = {
    'male': 'masculino',
    'female': 'feminino',
    'unknown': 'desconhecido' # Incluindo o valor 'unknown' (desconhecido) por s
}

# Aplicar o mapeamento de tradução à coluna 'sexo'
metadata['sexo'] = metadata['sexo'].replace(sexo_mapping)

print('Registros carregados:', len(metadata))
metadata.head()
```

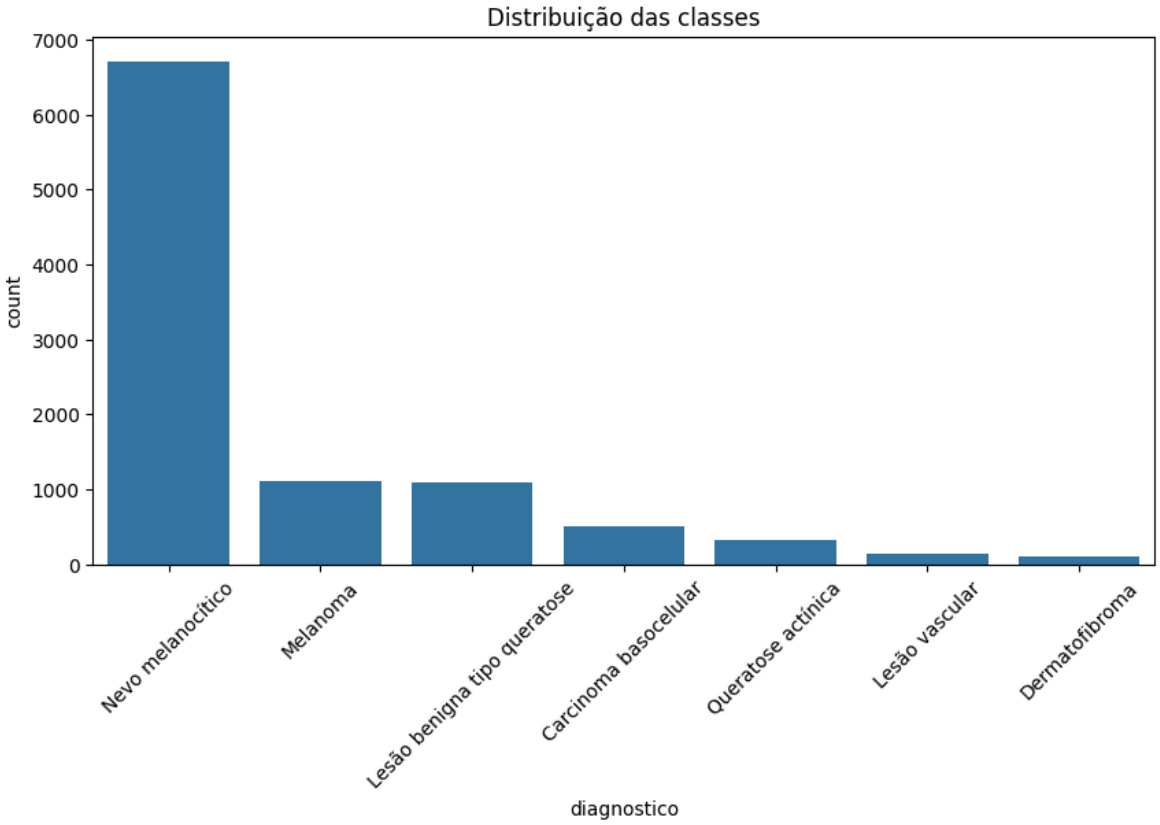
```
Registros carregados: 10015
```

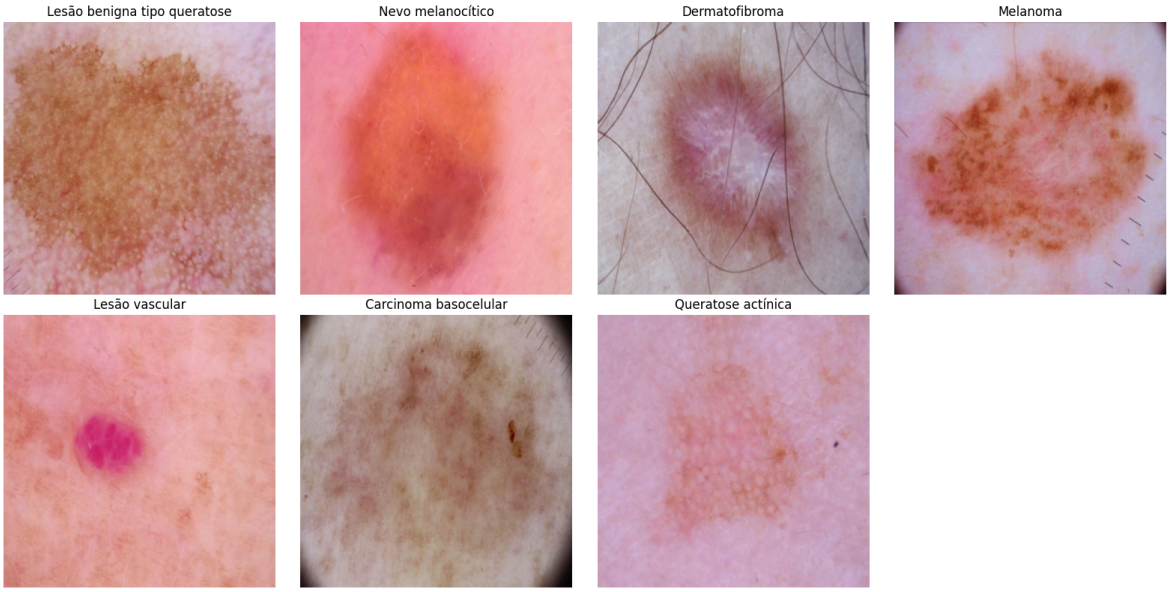
Out[4]:

	id_lesao	image_id	codigo_diagnostico	tipo_diagnostico	idade	sexo
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	masculino
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	masculino
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	masculino
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	masculino
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	masculino

🔍 Exploração de Dados

In [5]:
```
plt.figure(figsize=(10,5))
sns.countplot(data=metadata, x='diagnostico', order=metadata['diagnostico'].valu
plt.xticks(rotation=45)
plt.title('Distribuição das classes')
plt.show()

unique_labels = metadata['diagnostico'].unique()
cols = 4
rows = (len(unique_labels) + cols - 1) // cols
plt.figure(figsize=(4*cols, 4*rows))
for i, label in enumerate(unique_labels):
    subset = metadata[metadata['diagnostico'] == label]
    if subset.empty:
        continue
    sample_row = subset.sample(1, random_state=42).iloc[0]
    img = Image.open(sample_row['image_path']).convert('RGB')
    plt.subplot(rows, cols, i+1)
    plt.imshow(img.resize((256,256)))
    plt.title(label)
    plt.axis('off')
plt.tight_layout()
```





🧠 Criação e Treinamento do Modelo

In [6]:
```
le = LabelEncoder()
metadata['label'] = le.fit_transform(metadata['diagnostico'])
train_df, temp_df = train_test_split(metadata, stratify=metadata['label'], test_
val_df, test_df = train_test_split(temp_df, stratify=temp_df['label'], test_size

IMG_SIZE = (224,224)
BATCH_SIZE = 32

train_datagen = ImageDataGenerator(rescale=1./255, rotation_range=20, width_shif
val_datagen = ImageDataGenerator(rescale=1./255)

train_gen = train_datagen.flow_from_dataframe(train_df, x_col='image_path', y_co
val_gen = val_datagen.flow_from_dataframe(val_df, x_col='image_path', y_col='dia
test_gen = val_datagen.flow_from_dataframe(test_df, x_col='image_path', y_col='d
```

```
Found 7511 validated image filenames belonging to 7 classes.
Found 1252 validated image filenames belonging to 7 classes.
Found 1252 validated image filenames belonging to 7 classes.
```

In [7]:
```
base = MobileNetV2(weights='imagenet', include_top=False, input_shape=(IMG_SIZE[
for layer in base.layers:
    layer.trainable = False

x = GlobalAveragePooling2D()(base.output)
x = Dropout(0.3)(x)
num_classes = len(train_gen.class_indices)
outputs = Dense(num_classes, activation='softmax')(x)
model = Model(inputs=base.input, outputs=outputs)
model.compile(optimizer=Adam(1e-4), loss='categorical_crossentropy', metrics=['a
#model.summary()
```

In [8]:
```
### TREINANDO EPOCHS

# Obtém o nome das classes do gerador de treino
classes = list(train_gen.class_indices.keys())

# Exibe o número de amostras de validação por classe (útil para verificar balanc
val_counts = pd.Series(val_gen.classes).value_counts().reindex(range(len(classes
```

```
val_counts.index = classes
print("Distribuição das classes no conjunto de validação:\n")
print(val_counts)

# Calcula pesos de classe para lidar com desbalanceamento
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(train_gen.classes),
    y=train_gen.classes
)

# Converte o resultado em dicionário no formato aceito pelo Keras
class_weights = dict(enumerate(class_weights))
print("\nPesos de classe calculados:", class_weights)

# Treinamento do modelo
EPOCHS = 50
history = model.fit(
    train_gen,
    validation_data=val_gen,
    epochs=EPOCHS,
    class_weight=class_weights,
    verbose=0  # mostra o progresso do treinamento
)
```

```
Distribuição das classes no conjunto de validação:

Carcinoma basocelular            64
Dermatofibroma                   15
Lesão benigna tipo queratose    137
Lesão vascular                   18
Melanoma                        139
Nevo melanocítico               838
Queratose actínica               41
Name: count, dtype: int64

Pesos de classe calculados: {0: np.float64(2.787012987012987), 1: np.float64(12.4
76744186046512), 2: np.float64(1.3021844660194175), 3: np.float64(10.028037383177
57), 4: np.float64(1.2850299401197605), 5: np.float64(0.21336249751441638), 6: n
p.float64(4.3795918367346935)}
```

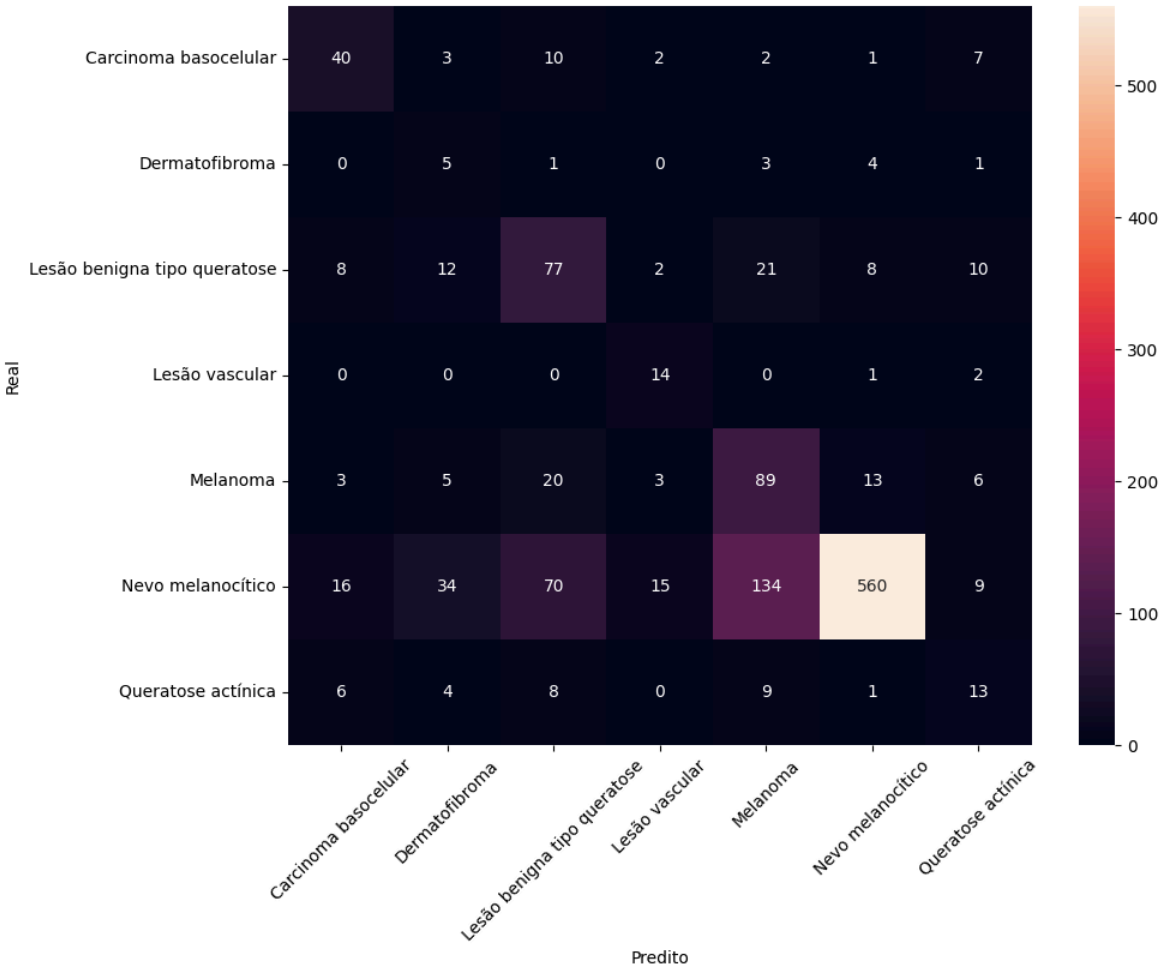
📖 Avaliação

In [9]:

```
preds = model.predict(test_gen)
y_pred = np.argmax(preds, axis=1)
y_true = test_gen.classes

print(classification_report(
    y_true,
    y_pred,
    target_names=list(train_gen.class_indices.keys()),
    zero_division=0
))

cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10,8))
ax = sns.heatmap(cm, annot=True, fmt='d', xticklabels=list(train_gen.class_indic
ax.set_xlabel('Predito')
ax.set_ylabel('Real')
```

```
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.show()
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 8s 178ms/step
                                precision    recall  f1-score   support

       Carcinoma basocelular       0.55      0.62      0.58        65
             Dermatofibroma        0.08      0.36      0.13        14
Lesão benigna tipo queratose       0.41      0.56      0.48       138
             Lesão vascular        0.39      0.82      0.53        17
                   Melanoma        0.34      0.64      0.45       139
          Nevo melanocítico        0.95      0.67      0.79       838
          Queratose actínica       0.27      0.32      0.29        41

                   accuracy                           0.64      1252
                  macro avg       0.43      0.57      0.46      1252
               weighted avg       0.76      0.64      0.68      1252
```



🔥 Funções Grad-CAM

In [26]:
```
### 🔍 Função para encontrar a última camada convolucional
def encontrar_ultima_conv(model):
    """
    Encontra automaticamente o nome da última camada convolucional
    (Conv2D, DepthwiseConv2D ou SeparableConv2D), mesmo em modelos aninhados.
    Retorna o nome da camada ou None se não encontrar.
    """
    ultima_conv = None
```

```
    def procurar_camadas(m):
        nonlocal ultima_conv
        for layer in m.layers:
            # Se for uma subcamada (ex: bloco dentro de EfficientNet)
            if hasattr(layer, 'layers') and len(layer.layers) > 0:
                procurar_camadas(layer)
            # Verifica se é uma camada convolucional
            if isinstance(layer, (tf.keras.layers.Conv2D,
                                  tf.keras.layers.DepthwiseConv2D,
                                  tf.keras.layers.SeparableConv2D)):
                ultima_conv = layer.name

    procurar_camadas(model)

    if ultima_conv is None:
        print("⚠️ Nenhuma camada convolucional encontrada. Modelo pode não ser C
    else:
        print(f"✅ Última camada convolucional encontrada: {ultima_conv}")

    return ultima_conv

ultima_conv = encontrar_ultima_conv(model)
print(ultima_conv)

def gerar_gradcam_heatmap(model, img_array, last_conv_layer_name, pred_index=Non
    grad_model = tf.keras.models.Model(model.input, [model.get_layer(last_conv_l
    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(predictions[0])
        class_channel = predictions[:, pred_index]
    grads = tape.gradient(class_channel, conv_outputs)
    pooled_grads = tf.reduce_mean(grads, axis=(0,1,2))

    conv_outputs = conv_outputs[0].numpy()
    pooled_grads = pooled_grads.numpy()
    for i in range(pooled_grads.shape[-1]):
        conv_outputs[:,:,i] *= pooled_grads[i]
    heatmap = np.mean(conv_outputs, axis=-1)
    heatmap = np.maximum(heatmap, 0)
    heatmap /= (heatmap.max() + 1e-8)
    heatmap = cv2.resize(heatmap, (IMG_SIZE[1], IMG_SIZE[0]))
    heatmap = np.uint8(255 * heatmap)
    heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
    return heatmap

def gerar_mapa_gradcam(model, imagem_array, nome_camada_conv):
    grad_model = tf.keras.models.Model(
        inputs=model.inputs,
        outputs=[model.get_layer(nome_camada_conv).output, model.output]
    )

    with tf.GradientTape() as tape:
        conv_output, pred = grad_model(imagem_array)
        classe_pred = tf.argmax(pred[0])
        loss = pred[:, classe_pred]

    grads = tape.gradient(loss, conv_output)[0]
    pesos = tf.reduce_mean(grads, axis=(0,1))
```

```
    gradcam = tf.reduce_sum(tf.multiply(pesos, conv_output[0]), axis=-1)

    heatmap = np.maximum(gradcam, 0) / (tf.reduce_max(gradcam) + 1e-8)
    heatmap = cv2.resize(heatmap.numpy(), (224,224))
    heatmap = np.uint8(255 * heatmap)
    heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
    return heatmap
```

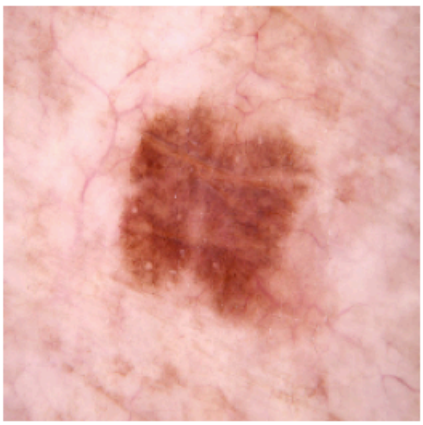
✅ Última camada convolucional encontrada: Conv_1
Conv_1

🖼️ Visualização Grad-CAM em Lote

In [27]:
```
def gradcam_em_lote(model, df, ultima_conv, IMG_SIZE=(224,224), num_imagens=6):

    exemplos = df.sample(num_imagens, random_state=42).reset_index(drop=True)
    plt.figure(figsize=(10, num_imagens * 3))

    for i, row in exemplos.iterrows():
        #  🔹 Carrega e normaliza imagem
        img_path = row['image_path']
        imagem = load_img(img_path, target_size=IMG_SIZE)
        imagem_arr = img_to_array(imagem) / 255.0
        entrada = tf.convert_to_tensor(np.expand_dims(imagem_arr, axis=0), dtype

        #  🔹 Gera mapa Grad-CAM
        mapa = gerar_mapa_gradcam(model, entrada, ultima_conv)

        #  🔹 Normaliza o mapa de calor para sobrepor
        mapa = cv2.resize(mapa, (IMG_SIZE[1], IMG_SIZE[0]))
        mapa = np.uint8(255 * mapa)
        mapa = cv2.applyColorMap(mapa, cv2.COLORMAP_JET)
        sobreposta = cv2.addWeighted(np.uint8(imagem_arr*255), 0.6, mapa, 0.4, 0

        #  🔹 Mostra imagem original
        plt.subplot(num_imagens, 2, 2*i + 1)
        plt.imshow(imagem)
        plt.title(f'Imagem Original: {row["diagnostico"]}')
        plt.axis('off')

        #  🔹 Mostra Grad-CAM
        plt.subplot(num_imagens, 2, 2*i + 2)
        plt.imshow(cv2.cvtColor(sobreposta, cv2.COLOR_BGR2RGB))
        plt.title('Mapa Grad-CAM')
        plt.axis('off')

    plt.tight_layout()
    plt.show()
```

In [28]:
```
gradcam_em_lote(model, test_df, ultima_conv, num_imagens=6)
```

Imagem Original: Queratose actínica



Mapa Grad-CAM

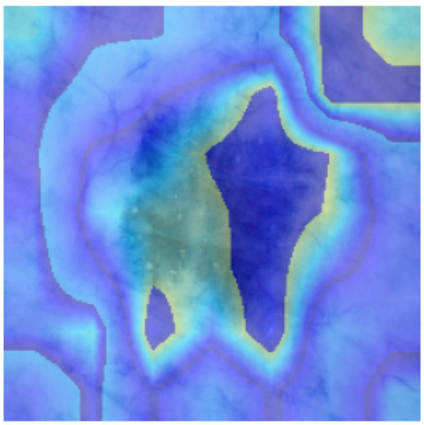
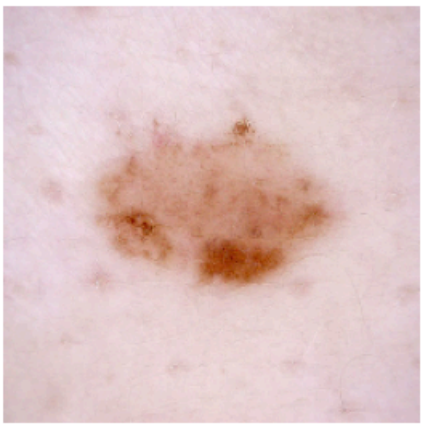


Imagem Original: Nevo melanocítico



Mapa Grad-CAM

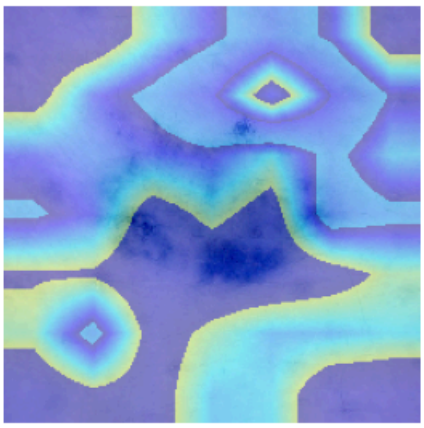


Imagem Original: Carcinoma basocelular

Mapa Grad-CAM

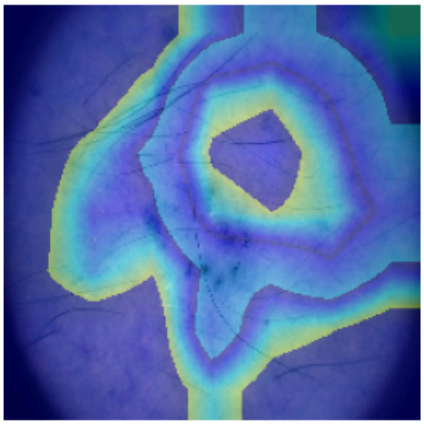
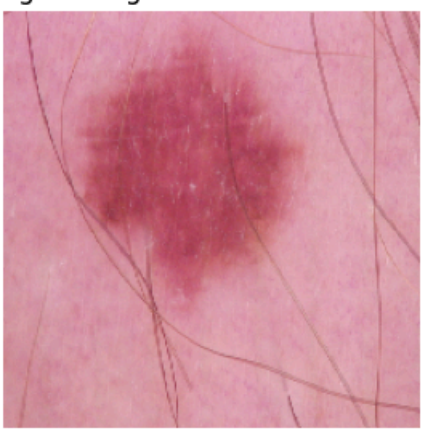


Imagem Original: Nevo melanocítico



Mapa Grad-CAM

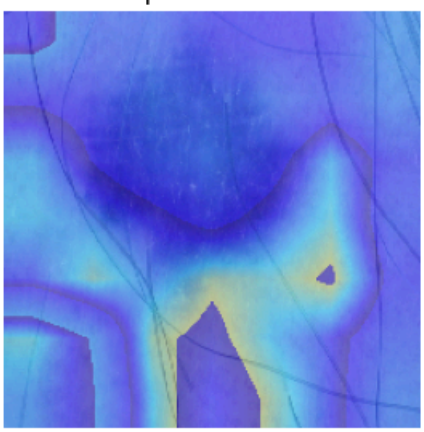
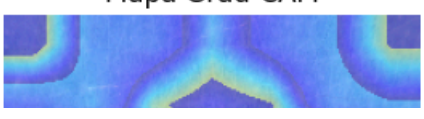
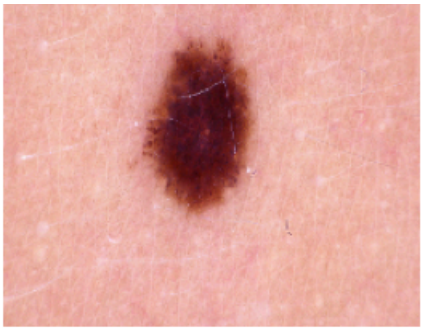


Imagem Original: Nevo melanocítico

Mapa Grad-CAM





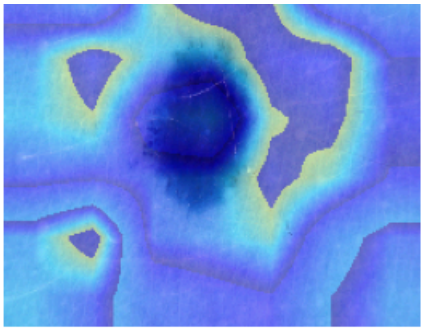
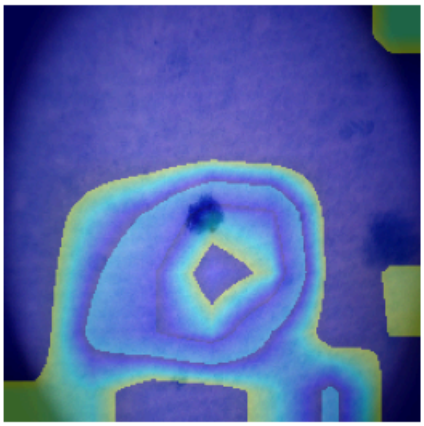


Imagem Original: Nevo melanocítico

Mapa Grad-CAM



💾 Salvamento do Modelo (formato moderno .keras)

In [29]:
```
model.save('modelo_cancer_pele.keras')
print('Modelo salvo com sucesso!')
```

```
Modelo salvo com sucesso!
```