📖 Tech Challenge: Visão Computacional

Desafio: Treinar um modelo para identificação de câncer de pele por imagens.

✅ 1. Importação das Bibliotecas

Abaixo iremos realizar o import das bibliotecas, e a definição do pytorch e do uso da GPU para visão computacional.

In [1]:
```
#--------------------------------------------------------------------------
# Imports Iniciais
#--------------------------------------------------------------------------
import os
from pathlib import Path
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
import keras
from keras import layers
from keras import ops
import torch
import cv2
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers, callbacks, applications
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
import torch, torch.nn as nn, torch.optim as optim
from torch.utils.data import DataLoader
import torchvision.transforms as T
from tensorflow.keras.applications import MobileNetV2
import kagglehub
from scipy import stats
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, LabelEncoder
from torch.utils.data import Dataset, DataLoader
from tqdm.auto import tqdm
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, i
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc
import warnings
from sklearn.utils import class_weight
from tensorflow.keras.optimizers import Adam
from sklearn.utils.class_weight import compute_class_weight

warnings.filterwarnings("ignore", category=UserWarning, module="keras.*trainers.
warnings.filterwarnings("ignore", category=UserWarning, module='tqdm')
os.environ["KERAS_BACKEND"] = "torch"
tf.keras.backend.set_image_data_format('channels_last')
tf.get_logger().setLevel('ERROR')

from keras import layers, Model, Input
from sklearn.model_selection import train_test_split
```

🧪 2. Preparação dos dados

Iremos iniciar aqui o download do primeiro dataset: kfarjanakabirsamanta/skin-cancer-dataset

Usaremos esse dataset para realizar o treinamento do modelo CNN

O dataset rm1000/skin-cancer-isic-images possui apenas imagens, sem rotulação por meio de um metadata. Usaremos essas imagens ao final para aplicação do modelo treinado.

In [2]:
```
#-------------------------------------------------------------------------
# Função para download dos datasets
#-------------------------------------------------------------------------
def download_dataset(base_dir_name):
    path = kagglehub.dataset_download(base_dir_name)
    print("Path to dataset files:", path)
    return path
```

In [3]:
```
#-------------------------------------------------------------------------
# Download dos Datasets: skin-cancer-dataset e skin-cancer-mnist-ham10000
#-------------------------------------------------------------------------
#dataset_file1 = download_dataset("kmader/skin-cancer-mnist-ham10000")
dataset_file1 = download_dataset("farjanakabirsamanta/skin-cancer-dataset")
dataset_file2 = download_dataset("rm1000/skin-cancer-isic-images")
```

```
Path to dataset files: C:\Users\flmli\.cache\kagglehub\datasets\farjanakabirsamanta\skin-cancer-dataset\versions\1
Path to dataset files: C:\Users\flmli\.cache\kagglehub\datasets\rm1000\skin-cancer-isic-images\versions\1
```

In [4]:
```
#-------------------------------------------------------------------------
# Carregando os arquivos dos metadados
#-------------------------------------------------------------------------
metadata = pd.read_csv(dataset_file1 + "/HAM10000_metadata.csv")

#-------------------------------------------------------------------------
# Validando datasets antes do tratamento dos dados
#-------------------------------------------------------------------------
print("Dataset 1: skin-cancer-dataset")
metadata.head()
```

```
Dataset 1: skin-cancer-dataset
```

Out[4]:

	lesion_id	image_id	dx	dx_type	age	sex	localization
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear

In [5]:
```
metadata.shape
metadata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10015 entries, 0 to 10014
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   lesion_id     10015 non-null  object 
 1   image_id      10015 non-null  object 
 2   dx            10015 non-null  object 
 3   dx_type       10015 non-null  object 
 4   age           9958 non-null   float64
 5   sex           10015 non-null  object 
 6   localization  10015 non-null  object 
dtypes: float64(1), object(6)
memory usage: 547.8+ KB
```

Iremos aqui definir o metadata, editar e criar os campos necessários para a análise exploratória e treinamento do modelo.

In [6]:
```
#-------------------------------------------------------------------------------
# Tratamento e preparação dos dados
#-------------------------------------------------------------------------------
mapeamento_diagnostico = {
    'akiec': 'Queratose actínica',
    'bcc'  : 'Carcinoma basocelular',
    'bkl'  : 'Lesão benigna tipo queratose',
    'df'   : 'Dermatofibroma',
    'mel'  : 'Melanoma',
    'nv'   : 'Nevo melanocítico',
    'vasc' : 'Lesão vascular'
}

metadata['diagnosis_desc'] = metadata['dx'].map(mapeamento_diagnostico)

mapeamento_localizacao = {
    'abdomen':          'Abdômen',
    'acral':            'Acrol',
    'back':             'Costas',
    'chest':            'Tórax',
    'ear':              'Orelha',
    'face':             'Face',
    'foot':             'Pé',
    'genital':          'Genitais',
    'hand':             'Mão',
    'lower extremity':  'Extremidade inferior',
    'neck':             'Pescoço',
    'scalp':            'Couro cabeludo',
    'trunk':            'Tronco',
    'unknown':          'Desconhecido',
    'upper extremity':  'Extremidade superior'
}

metadata['localization'] = metadata['localization'].map(mapeamento_localizacao).

renomean_col_mapeamento = {
    'lesion_id': 'id_lesao',
    'image_id': 'image_id',
    'dx': 'codigo_diagnostico',
    'dx_type': 'tipo_diagnostico',
    'age': 'idade',
    'sex': 'sexo',
```

```
    'localization': 'localizacao',
    'diagnosis_desc': 'diagnostico',
    'image_path': 'image_path'
}
metadata = metadata.rename(columns=renomean_col_mapeamento)

mapeamento_genero = {
    'male': 'Masculino',
    'female': 'Feminino',
    'unknown': 'Desconhecido'
}

metadata['sexo'] = metadata['sexo'].replace(mapeamento_genero)

mapeamento_tipo = {
    "histo": "Biópsia / Exame de tecido",
    "consensus": "Diagnóstico por consenso médico",
    "confocal": "Imagem confocal",
    "follow_up": "Acompanhamento clínico"
}

metadata['tipo_diagnostico'] = metadata['tipo_diagnostico'].replace(mapeamento_t

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

```
Registros carregados: 10015
```

In [7]:
```
#------------------------------------------------------------------------
# Adicionando caminho das imagens no dataframe
#------------------------------------------------------------------------
#PARTS   = [os.path.join(dataset_file1, 'HAM10000_images_part_1'),
#           os.path.join(dataset_file1, 'HAM10000_images_part_2')]

base_path = os.path.join(dataset_file1, "Skin Cancer", "Skin Cancer")

def encontrar_imagens(image_id):
    nome_arquivo = f"{image_id}.jpg"
    caminho = os.path.join(base_path, nome_arquivo)
    if os.path.isfile(caminho):
        return caminho
    return None

metadata['image_path'] = metadata['image_id'].apply(encontrar_imagens)
metadata['image_path'] = metadata['image_path'].astype(str)

num_encontradas = metadata['image_path'].apply(lambda x: os.path.isfile(x)).sum(
print(f"Imagens encontradas no dataset: {num_encontradas} / {len(metadata)}")
```

```
Imagens encontradas no dataset: 10015 / 10015
```

Aqui criamos uma coluna binária para definição de positivo ou negativo para câncer

Por definição, é considerado câncer de pele quando o diagnóstico possui um desses 3 valores: Melanoma (mel), Carcinoma basocelular (bcc) e Queratose actínica (akiec).

In [8]:
```
#------------------------------------------------------------------------
# Criando coluna indicando câncer ou não para análise exploratória
#------------------------------------------------------------------------
codigos_cancer = {'akiec', 'bcc', 'mel'}
```

```
metadata['cancer'] = metadata['codigo_diagnostico'].isin(codigos_cancer).astype(
metadata.head()
```

Out[8]:

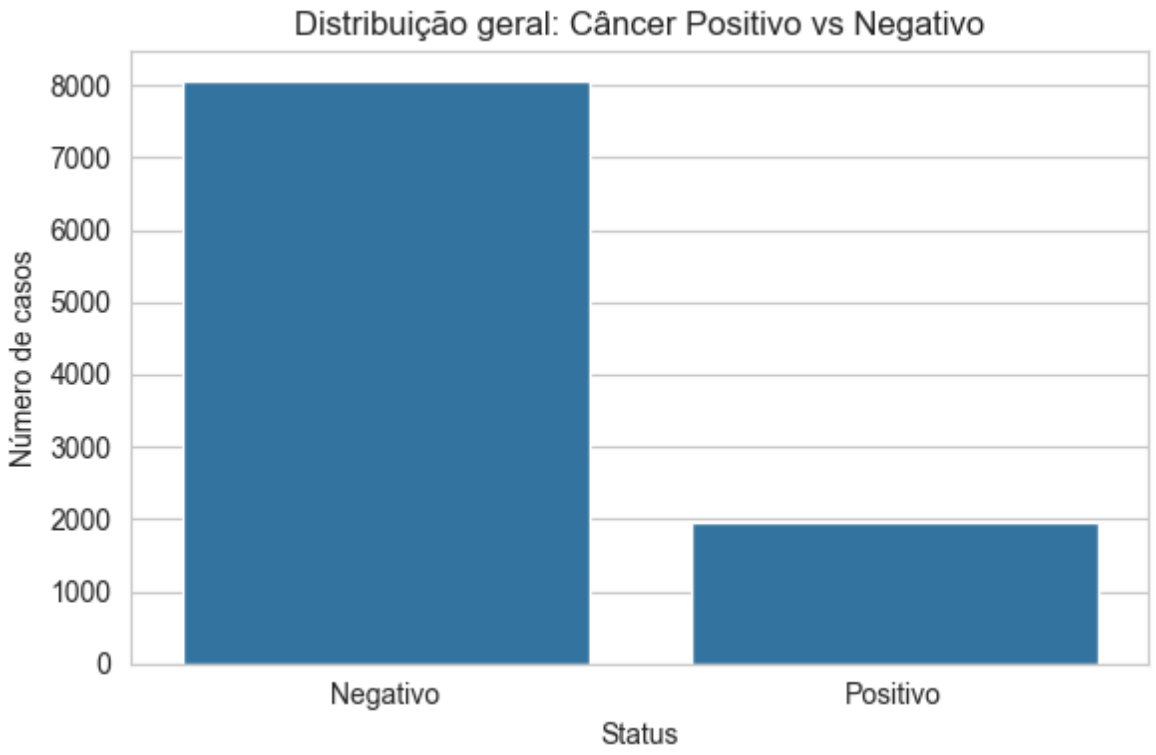
	id_lesao	image_id	codigo_diagnostico	tipo_diagnostico	idade	sexo
0	HAM_0000118	ISIC_0027419	bkl	Biópsia / Exame de tecido	80.0	Masculino
1	HAM_0000118	ISIC_0025030	bkl	Biópsia / Exame de tecido	80.0	Masculino
2	HAM_0002730	ISIC_0026769	bkl	Biópsia / Exame de tecido	80.0	Masculino
3	HAM_0002730	ISIC_0025661	bkl	Biópsia / Exame de tecido	80.0	Masculino
4	HAM_0001466	ISIC_0031633	bkl	Biópsia / Exame de tecido	75.0	Masculino

👀 3. Análise Exploratória

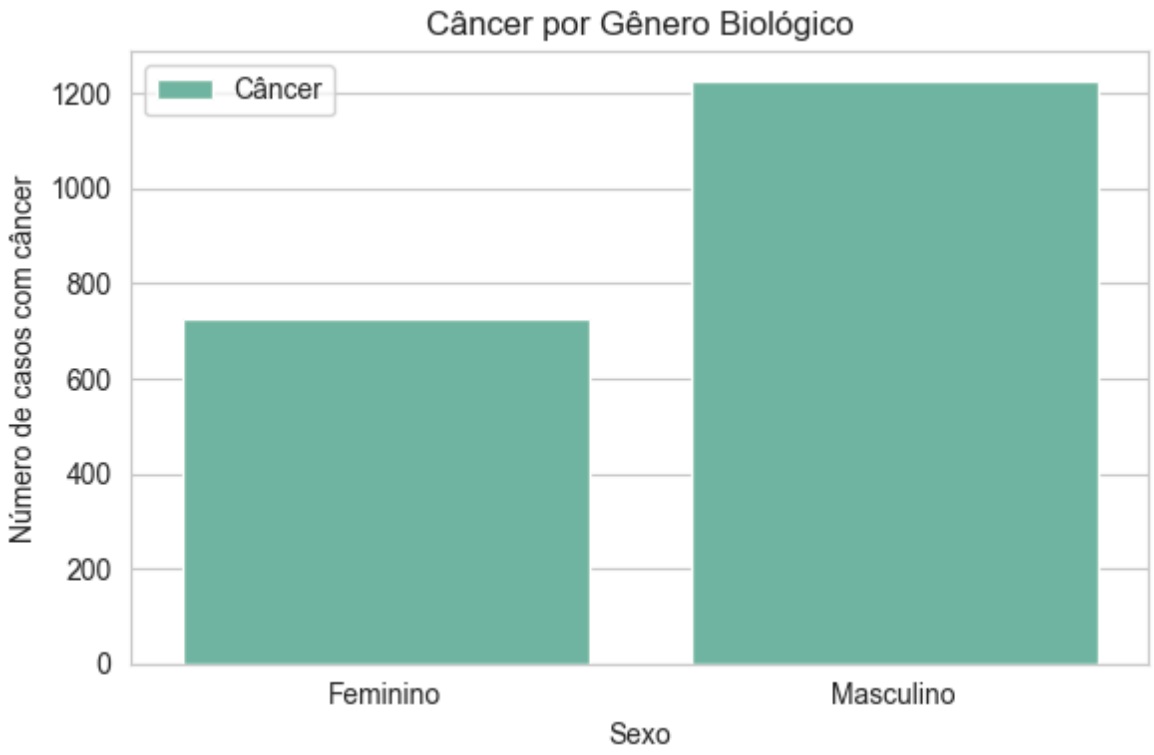
Aqui iremos extrair alguns gráficos para analisar os dados do dataframe antes de partirmos para a preparação do modelo CNN.

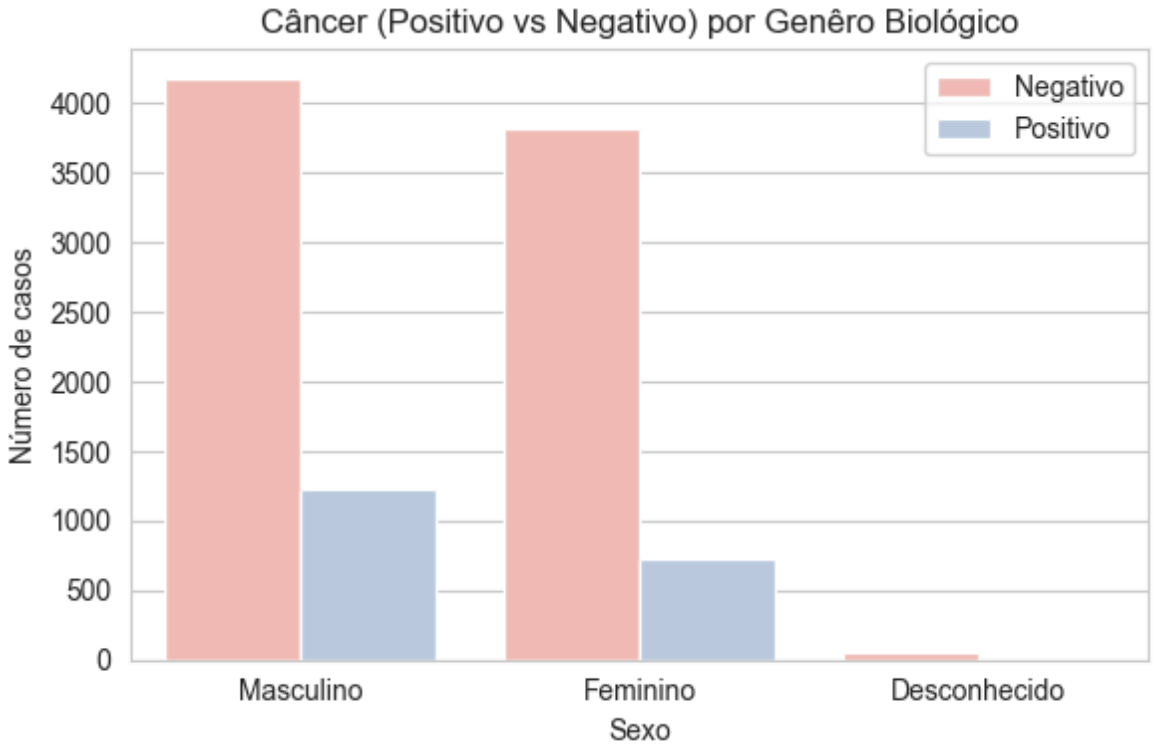
In [9]:

```
#-------------------------------------------------------------------------
# Análise Exploratória
#-------------------------------------------------------------------------
sns.set_style("whitegrid")

#-------------------------------------------------------------------------
# Gráfico de barras: Distribuição geral de câncer positivo vs negativo
#-------------------------------------------------------------------------
plt.figure(figsize=(6,4))
ax = sns.countplot(data=metadata, x='cancer')
ax.set_xticks([0, 1])
ax.set_xticklabels(['Negativo', 'Positivo'])
ax.set_xlabel('Status')
ax.set_ylabel('Número de casos')
ax.set_title('Distribuição geral: Câncer Positivo vs Negativo')
plt.tight_layout()
plt.show()
```



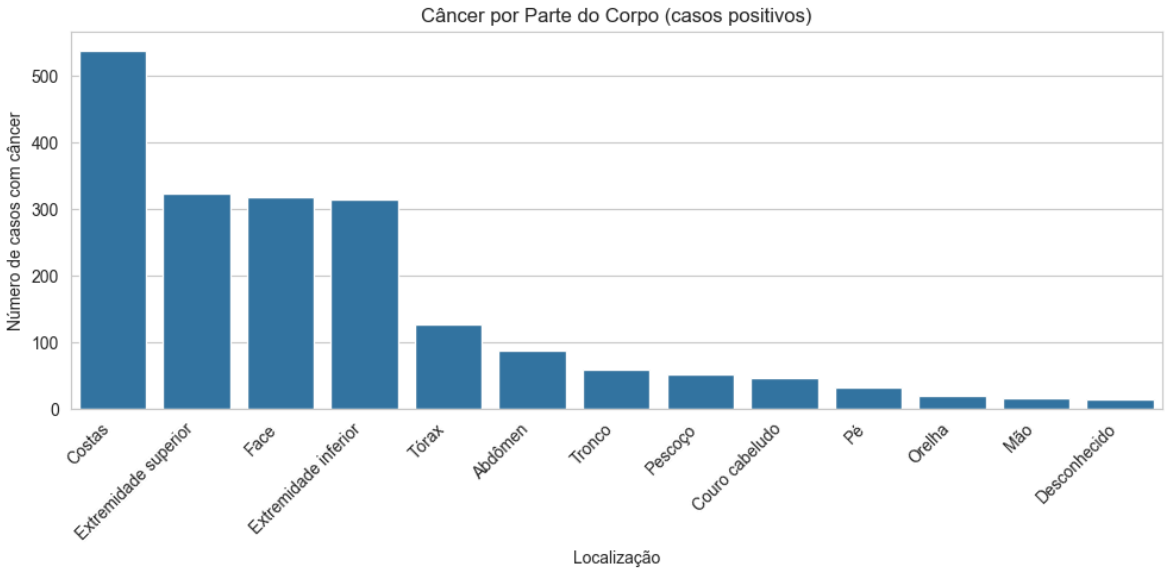
In [10]:
```
#-------------------------------------------------------------------------
# Gráfico de barras: Câncer por Gênero Biológico
#-------------------------------------------------------------------------
plt.figure(figsize=(6,4))
ax = sns.countplot(data=metadata[metadata['cancer']==1],
                   x='sexo', hue='cancer',
                   palette="Set2")
ax.set_xlabel('Sexo')
ax.set_ylabel('Número de casos com câncer')
ax.set_title('Câncer por Gênero Biológico')
plt.legend(title='', labels=['Câncer'])
plt.tight_layout()
plt.show()

fig, ax = plt.subplots(figsize=(6,4))
sns.countplot(data=metadata,
              x='sexo',
              hue='cancer',
              palette="Pastel1",
              ax=ax)
ax.set_xlabel('Sexo')
ax.set_ylabel('Número de casos')
ax.set_title('Câncer (Positivo vs Negativo) por Genêro Biológico')
plt.legend(title='', labels=['Negativo', 'Positivo'])
plt.tight_layout()
plt.show()
```

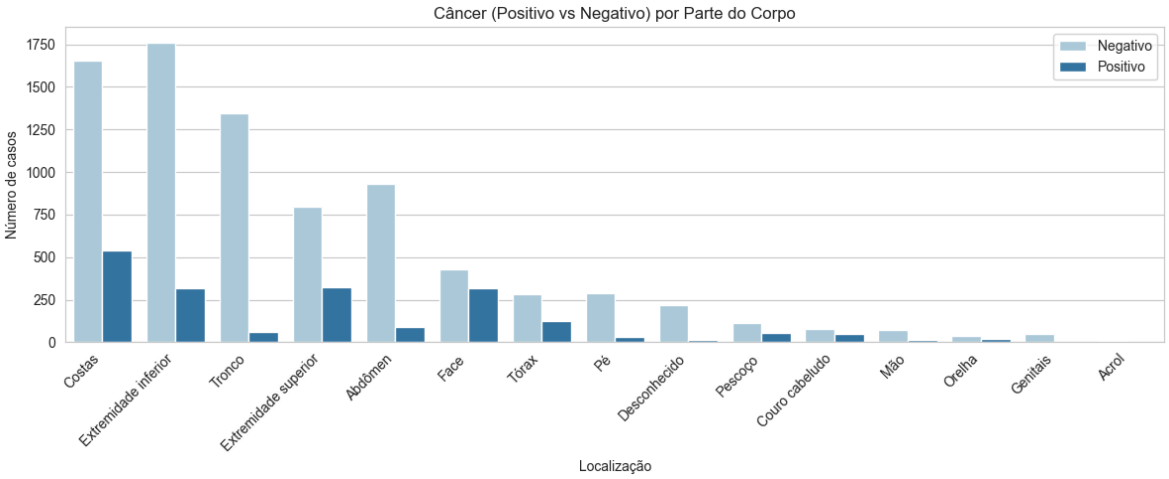




In [11]:
```
#-------------------------------------------------------------------------
# Gráfico de barras: Câncer por Localização
#-------------------------------------------------------------------------
locs = metadata[metadata['cancer']==1]['localizacao'].value_counts().index

plt.figure(figsize=(10,5))
ax = sns.countplot(data=metadata[metadata['cancer']==1],
                   x='localizacao',
                   order=locs)
ax.set_xlabel('Localização')
ax.set_ylabel('Número de casos com câncer')
ax.set_title('Câncer por Parte do Corpo (casos positivos)')
plt.xticks(rotation=45, ha='right')
```

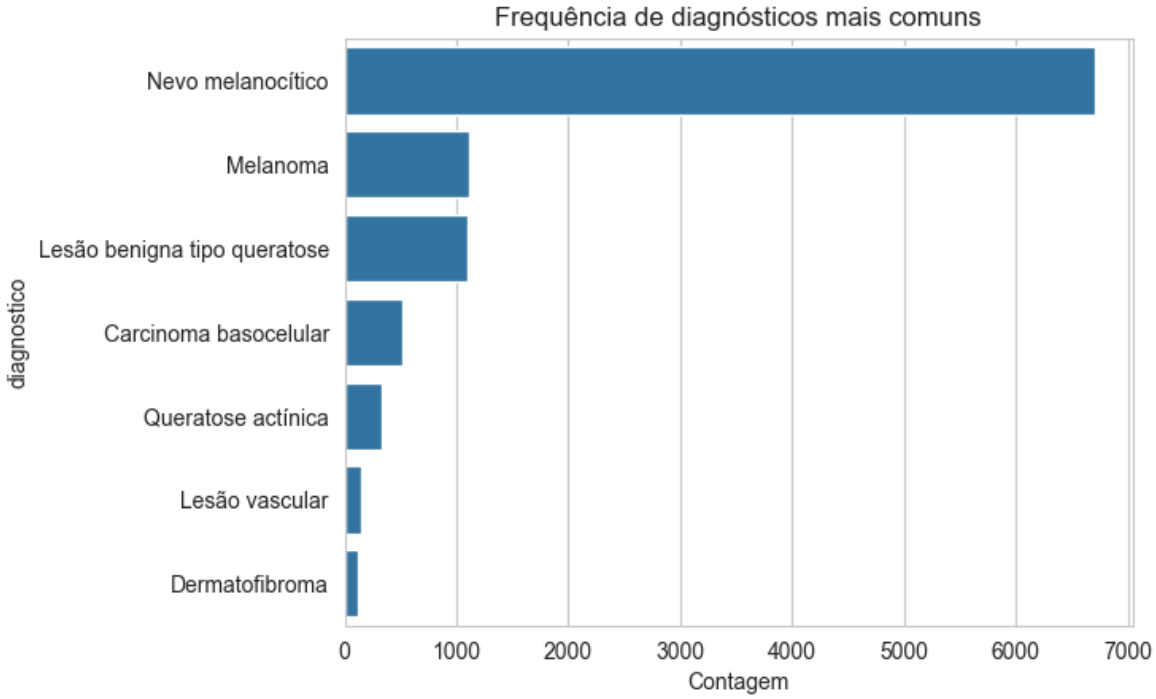
```
plt.tight_layout()
plt.show()

fig, ax = plt.subplots(figsize=(12,5))
sns.countplot(data=metadata,
              x='localizacao',
              hue='cancer',
              order=metadata['localizacao'].value_counts().index,
              palette="Paired",
              ax=ax)
ax.set_xlabel('Localização')
ax.set_ylabel('Número de casos')
ax.set_title('Câncer (Positivo vs Negativo) por Parte do Corpo')
plt.xticks(rotation=45, ha='right')
plt.legend(title='', labels=['Negativo', 'Positivo'])
plt.tight_layout()
plt.show()
```





In [12]:

```
#-------------------------------------------------------------------------
# Gráfico de diagnósticos mais frequentes
#-------------------------------------------------------------------------
top_diag = metadata['diagnostico'].value_counts().head(10)
sns.barplot(x=top_diag.values, y=top_diag.index)
plt.title('Frequência de diagnósticos mais comuns')
plt.xlabel('Contagem')
plt.show()
```



Aqui já percebemos que há um desbalanceamento de classes, com a classe "Nevo melanocítico" possuindo muito mais frequência que as demais classes. Isso deverá ser levado em conta na hora de definirmos o peso para cada classe no processamento do modelo.

Abaixo iremos extrair algumas amostras visuais de nosso dataset.

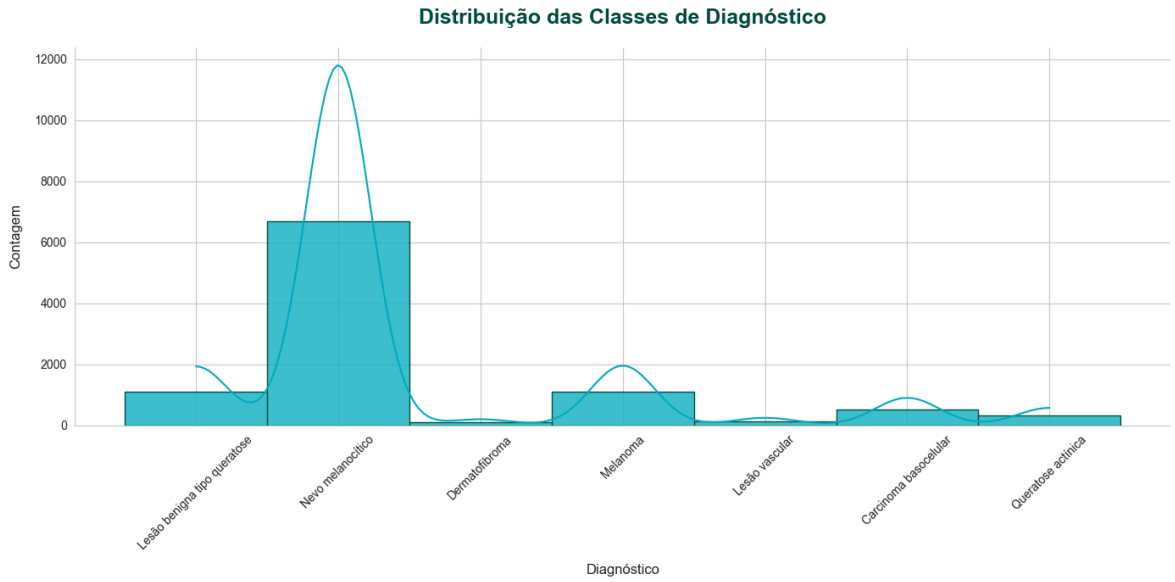
In [13]:
```
# ----------------------------------------------------------------
# Amostras Visuais
# ----------------------------------------------------------------
print("\nAmostras visuais por classe de diagnóstico:")

unique_labels = metadata['diagnostico'].unique()
cols = 4
rows = (len(unique_labels) + cols - 1) // cols

plt.figure(figsize=(3*cols, 3*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((128,128)))
    plt.title(label, fontsize=11, color="#004D40", weight='semibold')
    plt.axis('off')

plt.suptitle("Exemplos de Imagens por Classe", fontsize=20, weight='bold', color
plt.tight_layout()
plt.show()
```

```
Amostras visuais por classe de diagnóstico:
```

Vamos tirar mais alguns gráficos para analisar o balanceamento das classes.

In [14]:
```
# ----------------------------------------------------------------------
# Distribuição das Classes
# ----------------------------------------------------------------------
plt.figure(figsize=(14, 7))

sns.histplot(
    data=metadata,
    x='diagnostico',
    kde=True,
    bins=len(metadata['diagnostico'].unique()),
    color="#00ACC1",
    edgecolor="#004D40",
    alpha=0.75
)

print("Contagem por classe de diagnóstico:")

plt.title("Distribuição das Classes de Diagnóstico", fontsize=18, weight='bold',
plt.xlabel("Diagnóstico", fontsize=12, labelpad=10)
plt.ylabel("Contagem", fontsize=12, labelpad=10)
plt.xticks(rotation=45, fontsize=10)
plt.yticks(fontsize=10)
sns.despine()
plt.tight_layout()
plt.show()

contagem = metadata['diagnostico'].value_counts()
print(contagem)
```

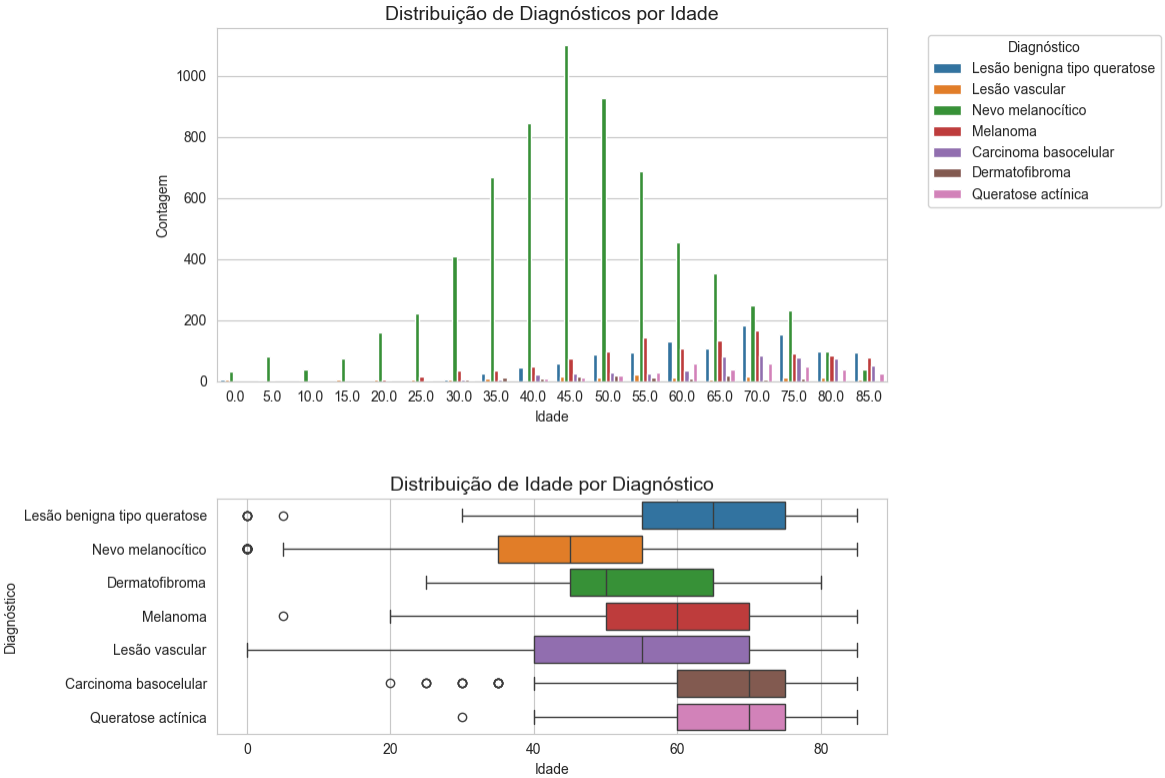
```
Contagem por classe de diagnóstico:
```




```
diagnostico
Nevo melanocítico              6705
Melanoma                       1113
Lesão benigna tipo queratose   1099
Carcinoma basocelular           514
Queratose actínica              327
Lesão vascular                  142
Dermatofibroma                  115
Name: count, dtype: int64
```

In [15]:
```
# ------------------------------------------------------------------
# Histograma: contagem de diagnósticos por idade
# ------------------------------------------------------------------
plt.rcParams["figure.figsize"] = (12, 8)
unique_diag = sorted(metadata['diagnostico'].unique())

fig, axes = plt.subplots(
    nrows=2,
    ncols=1,
    gridspec_kw={'height_ratios': [1.5, 1]},
    sharex=False
)

ax_hist, ax_box = axes

hist_obj = sns.countplot(
    data=metadata,
    x='idade',
    hue='diagnostico',
    ax=ax_hist
)

ax_hist.set_title('Distribuição de Diagnósticos por Idade', fontsize=14)
ax_hist.set_xlabel('Idade')
ax_hist.set_ylabel('Contagem')
hist_handles, hist_labels = hist_obj.get_legend_handles_labels()

ax_hist.legend(
    title='Diagnóstico',
    bbox_to_anchor=(1.05, 1),
    loc='upper left'
)
```

```
# ------------------------------------------------------------------
# Boxplot – distribuição de idade por diagnóstico (sem warning)
# ------------------------------------------------------------------
box_obj = sns.boxplot(
    data=metadata,
    x='idade',
    y='diagnostico',
    hue='diagnostico',
    legend=False,
    ax=ax_box
)

if ax_box.get_legend() is not None:
    ax_box.get_legend().remove()

ax_box.set_title('Distribuição de Idade por Diagnóstico', fontsize=14)
ax_box.set_xlabel('Idade')
ax_box.set_ylabel('Diagnóstico')

# ------------------------------------------------------------------
# Exibição dos gráficos
# ------------------------------------------------------------------
plt.tight_layout()
plt.subplots_adjust(hspace=0.4)
plt.show()
```

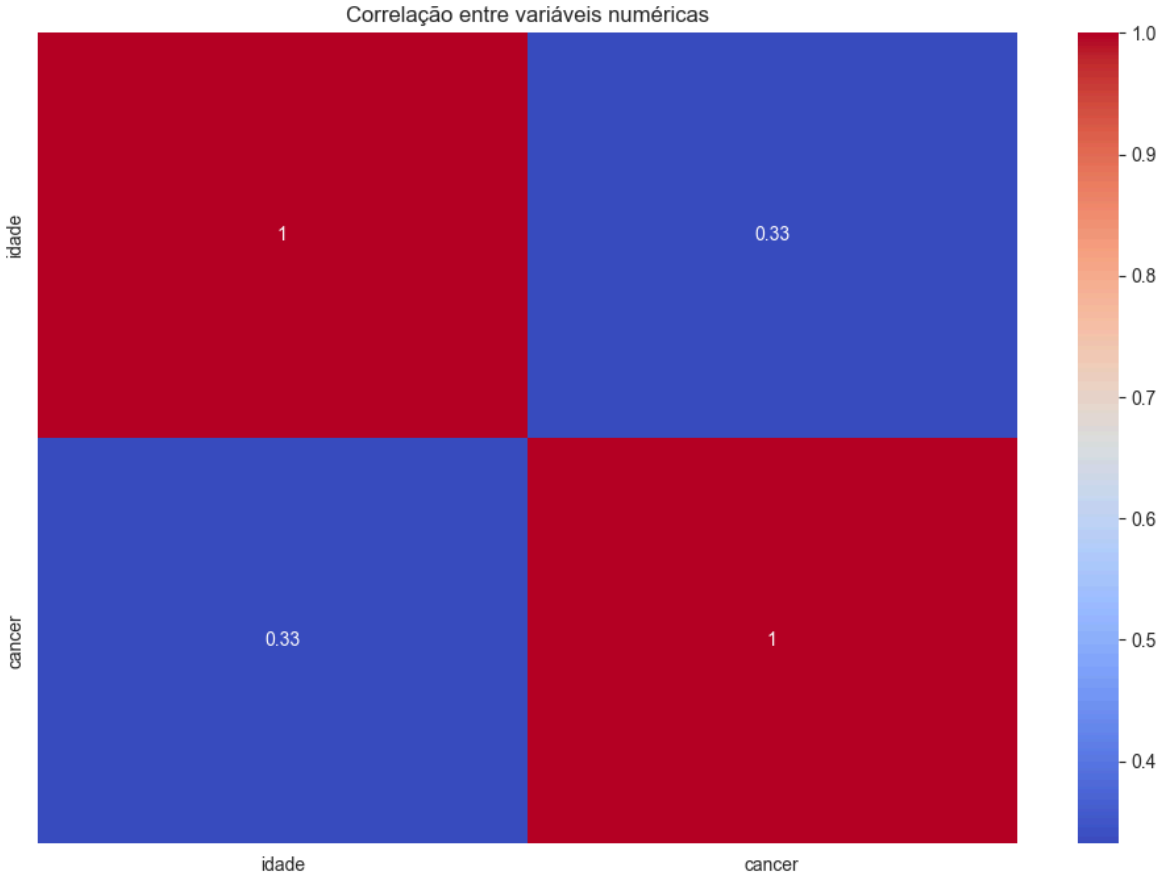


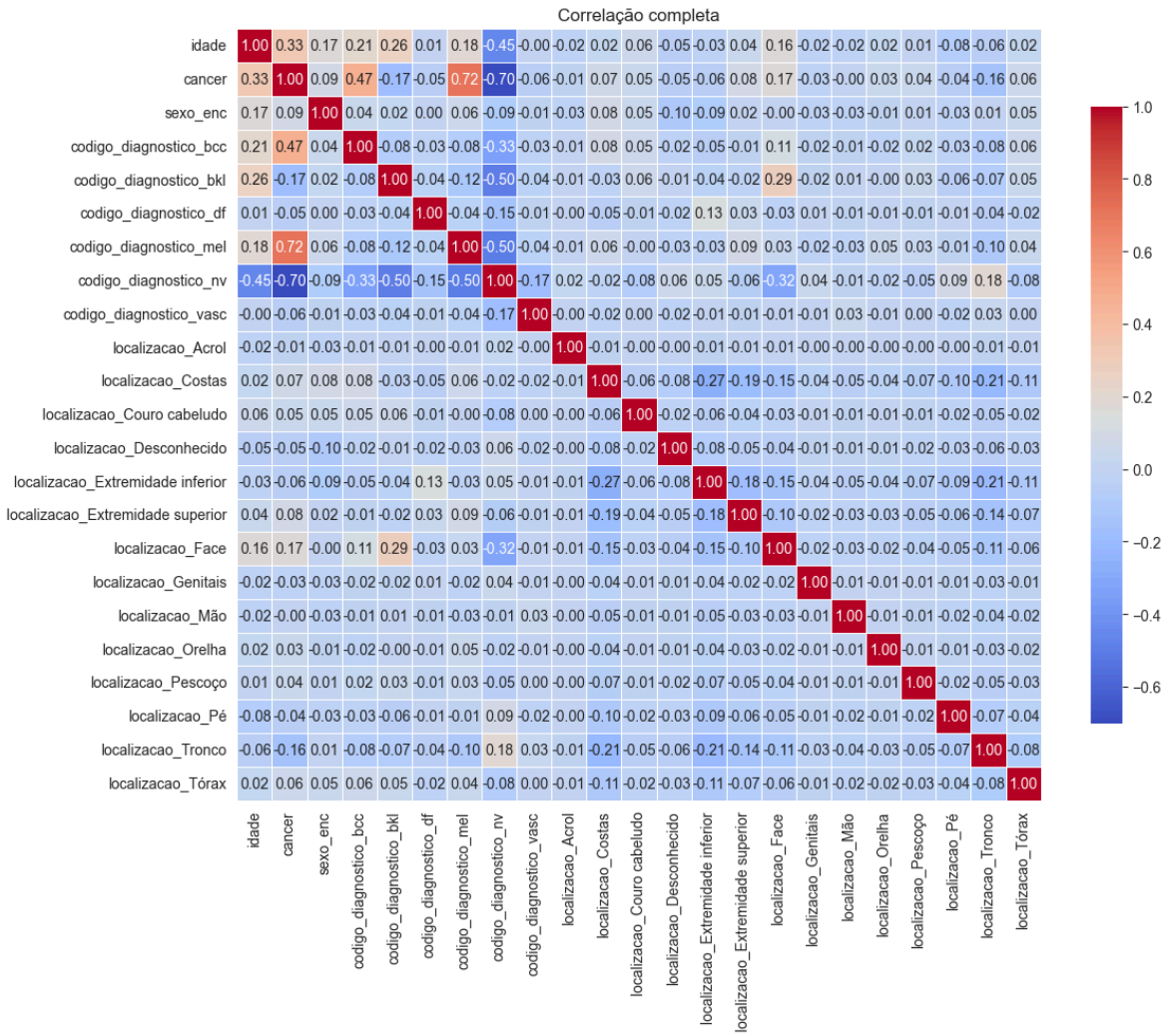
Vamos analisar a correlação entre as variáveis numéricas ou binárias

Depois iremos verificar a correlação das principais variáveis classificatórias

In [16]:
```
#-------------------------------------------------------------------------
# Correlação entre variáveis numéricas
#-------------------------------------------------------------------------
corr = metadata.select_dtypes(include=['number']).corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlação entre variáveis numéricas')
```

```
plt.show()

#-------------------------------------------------------------------------
# Correlação entre variáveis categóricas e numéricas
#-------------------------------------------------------------------------
metadata_corr = metadata.copy()
le_sex = OrdinalEncoder()
metadata_corr['sexo_enc'] = le_sex.fit_transform(metadata_corr[['sexo']])
cols_categoricas = ['codigo_diagnostico', 'localizacao']

oh = OneHotEncoder(drop='first',
                   sparse_output=False)
oh_df = pd.DataFrame(oh.fit_transform(metadata_corr[cols_categoricas]),
                     columns=oh.get_feature_names_out(cols_categoricas),
                     index=metadata_corr.index)

df_corr = metadata_corr.drop(columns=cols_categoricas + ['sexo']) \
             .join(oh_df)

cols_to_drop = ['id_lesao', 'image_id',
                'tipo_diagnostico', 'image_path','diagnostico']
df_corr_clean = df_corr.drop(columns=cols_to_drop)

corr2 = df_corr_clean.corr()

plt.figure(figsize=(12,10))
sns.heatmap(corr2, annot=True, cmap='coolwarm', fmt='.2f',
            linewidths=.5, cbar_kws={'shrink': .8})
plt.title('Correlação completa')
plt.tight_layout()
plt.show()
```

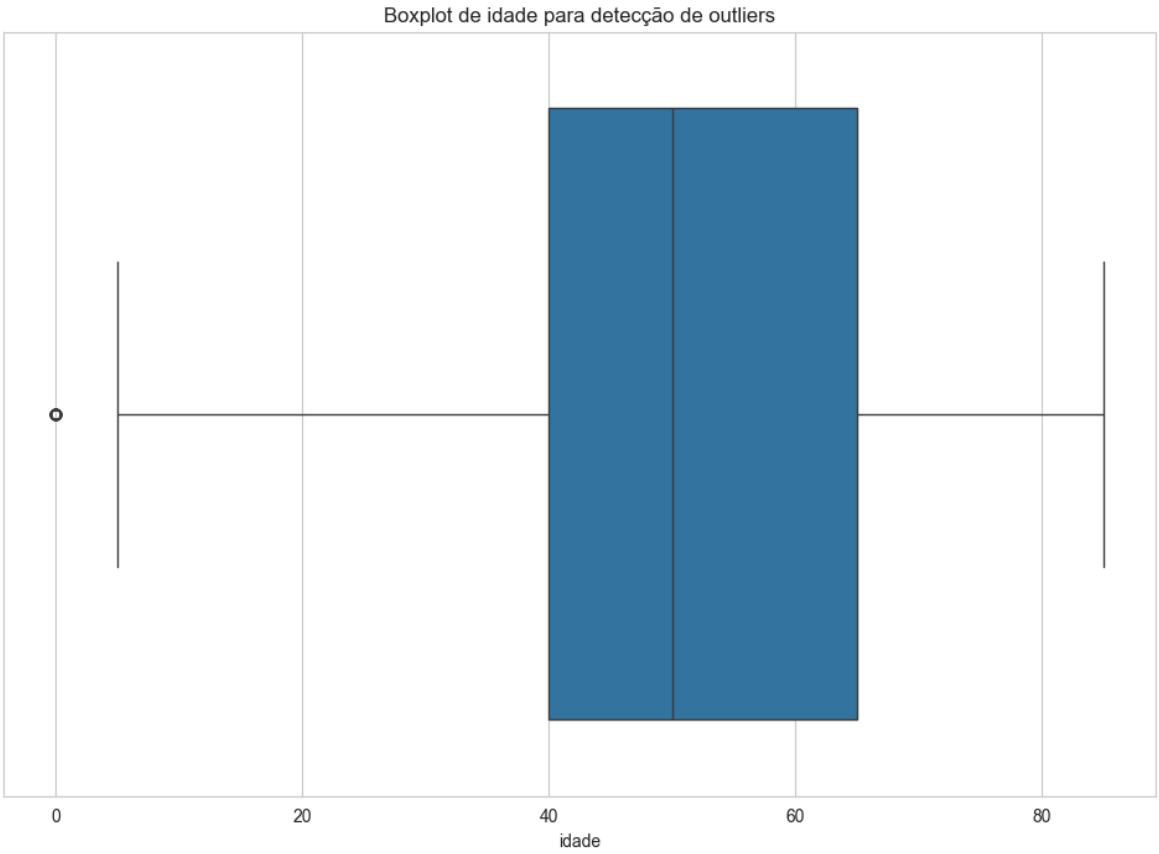




Vamos detectar os outliers na única coluna numérica que temos: idade

In [17]:
```
#-------------------------------------------------------------------------------
# Detectando outliers
#-------------------------------------------------------------------------------
z_scores = np.abs(stats.zscore(metadata["idade"].dropna()))
threshold = 3
outliers_mask_z = z_scores > threshold
print(f"Número de outliers (Z-score) em idade:", outliers_mask_z.sum())

sns.boxplot(x=metadata["idade"])
plt.title(f'Boxplot de {"idade"} para detecção de outliers')
plt.show()
```

```
Número de outliers (Z-score) em idade: 39
```



🧹 4. Limpeza de Dados

Aqui iremos realizar uma limpeza de dados antes do processamento do modelo.

Identificamos pelos gráficos que necessitaremos excluir da base os registros com a informação aonde o genêro retornou "desconhecido" e a localização retornou "desconhecido".

Também iremos remover os outliers do campo idade.

In [18]:
```
#-------------------------------------------------------------------------
# Removendo outliers
#-------------------------------------------------------------------------
idx_outliers = metadata.index[metadata["idade"].notna()][outliers_mask_z]

metadata_outlier = metadata.drop(index=idx_outliers).reset_index(drop=True)
print(f"Linhas após remoção: {len(metadata_outlier)}")

metadata = metadata_outlier.copy()

#-------------------------------------------------------------------------
# Remoção de valores desconhecidos de "sexo" e "localizacao" do metadata
#-------------------------------------------------------------------------
print(f"Linhas antes de remover inválidos: {len(metadata)}")
metadata = metadata[metadata['sexo'].isin(['Masculino', 'Feminino'])]
metadata = metadata[metadata['localizacao'].notna()]
print(f"Linhas após remover inválidos: {len(metadata)}")
```

```
Linhas após remoção: 9976
Linhas antes de remover inválidos: 9976
Linhas após remover inválidos: 9921
```

In [19]:
```
#-------------------------------------------------------------------------
# Limpeza do Metadata
#-------------------------------------------------------------------------
metadata_limpo = metadata.copy()
metadata_limpo.drop_duplicates(inplace=True)
print(f"Linhas após remover duplicados: {len(metadata_limpo)}")

cols_cruciais = ['diagnostico', 'localizacao', 'cancer', 'idade', 'sexo', 'codig
metadata_limpo.dropna(subset=cols_cruciais, inplace=True)

if metadata_limpo['idade'].isna().any():
    metadata_limpo['idade'].fillna(metadata_limpo['idade'].median(), inplace=Tru

print(f"Linhas após tratar faltantes: {len(metadata_limpo)}")

le_diag = LabelEncoder()
metadata_limpo['diag_enc'] = le_diag.fit_transform(metadata_limpo['diagnostico']
metadata_limpo = metadata_limpo[metadata_limpo['image_path'].apply(os.path.exist
print(f"Linhas após filtrar arquivos inexistentes: {len(metadata_limpo)}")
```

```
Linhas após remover duplicados: 9921
Linhas após tratar faltantes: 9911
Linhas após filtrar arquivos inexistentes: 9911
```

Preparar Treinamento

Aqui iremos preparar os dados para o treinamento do modelo CNN.

Iremos criar os modelos de teste e de validação.

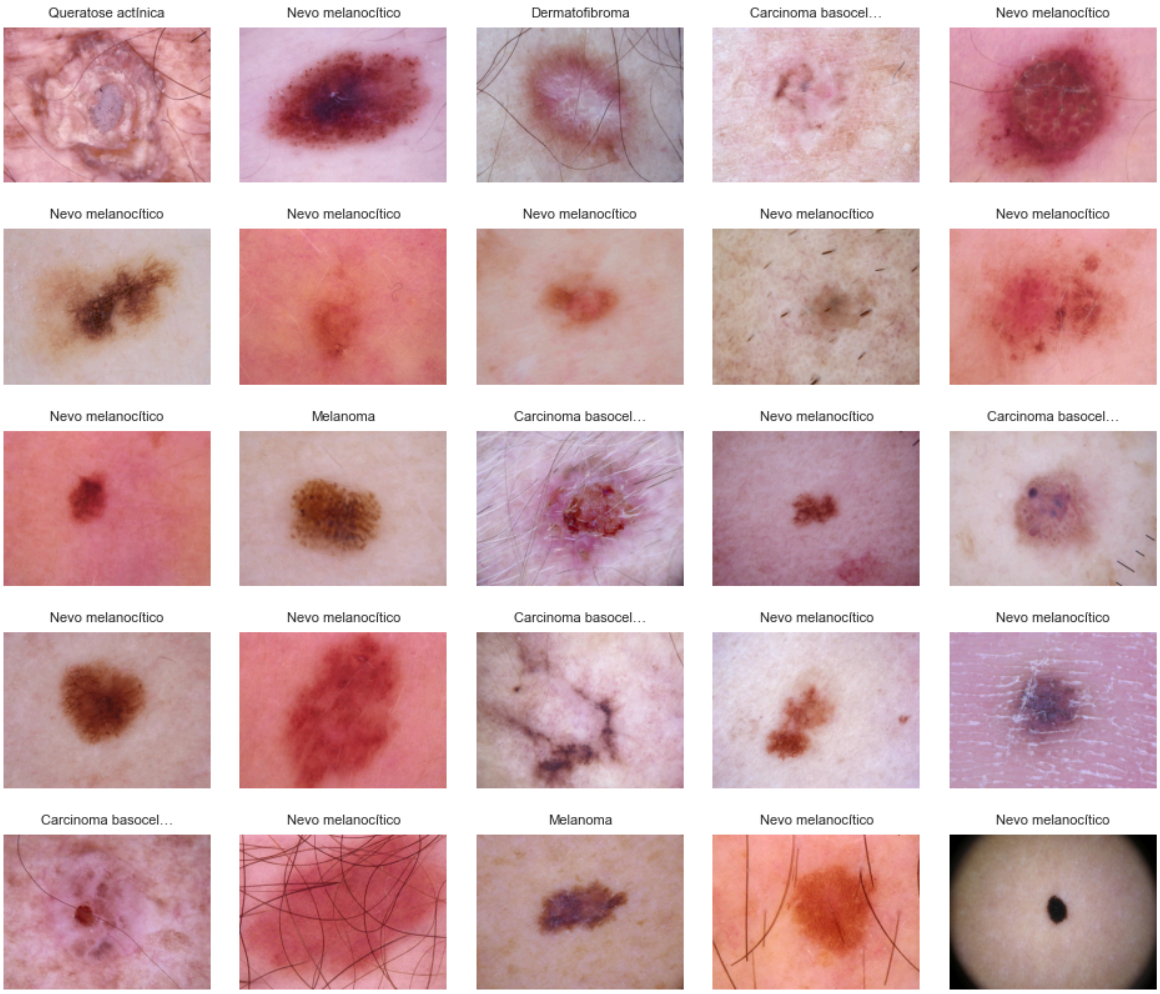
In [20]:
```
#-------------------------------------------------------------------------
# Preparando Treinamento
#-------------------------------------------------------------------------
le = LabelEncoder()
metadata['label'] = le.fit_transform(metadata['diagnostico'])
train_df, temp_df = train_test_split(metadata, 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 7440 validated image filenames belonging to 7 classes.
Found 1240 validated image filenames belonging to 7 classes.
Found 1241 validated image filenames belonging to 7 classes.
```

Abaixo iremos plotar no gráfico as primeiras 25 imagens do dataset de teste para validação do mesmo.

In [21]:
```
#-------------------------------------------------------------------------
# Plotando primeiras 25 imagens do conjunto de treinamento
#-------------------------------------------------------------------------
plt.figure(figsize=(12, 10))
for i in range(25):
```

```
    ax = plt.subplot(5, 5, i+1)
    img_path = train_df.iloc[i]['image_path']
    img      = Image.open(img_path)
    ax.imshow(img, cmap=plt.cm.binary)
    ax.set_xticks([])
    ax.set_yticks([])
    title_text = str(train_df.iloc[i]['diagnostico'])
    if len(title_text) > 20:
        title_text = title_text[:17] + '…'
    ax.set_title(title_text, fontsize=8, loc='center')
    ax.grid(False)
    ax.axis('off')
    plt.subplots_adjust(wspace=0.1, hspace=0.3)
plt.show()
```



Abaixo iremos preparar o modelo CNN

Optei por usar o backbone "MobileNetV2" após diversos testes por possuir a melhor performance e também a melhor precisão que consegui durante os testes.

Nos comentários está um código com um modelo CNN criado, porém que teve um desempenho muito abaixo do esperado.

In [22]:
```
# ---------------------------------------------------------------
# Executando o CNN: MobileNetV2
# ---------------------------------------------------------------
base = MobileNetV2(weights='imagenet', include_top=False, input_shape=(IMG_SIZE[

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)

for layer in base.layers[:-5]:
    layer.trainable = False

model.compile(optimizer=Adam(1e-4),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()

###### MODELO CNN CONSTRUIDO ANTERIORMENTE
###### TROQUEI ESTE POR UM MOBILENETV3 DEVIDO A DESEMPENHO E ACURACIA BAIXA

#IMG_HEIGHT, IMG_WIDTH = IMG_SIZE

#NUM_CLASSES = len(train_gen.class_indices)
#print(f"Número de diagnósticos: {NUM_CLASSES}")

#metrics=[tf.keras.metrics.CategoricalAccuracy(),
#         tf.keras.metrics.Precision(),
#         tf.keras.metrics.Recall()]
#l2 = 0.01

#model = models.Sequential()

#model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_sh
#model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same', kernel_r
#model.add(layers.MaxPooling2D((2, 2)))
#model.add(layers.Dropout(0.25))

#model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same', kernel_r
#model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same', kernel_r
#model.add(layers.MaxPooling2D((2, 2)))
#model.add(layers.Dropout(0.25))

#model.add(layers.GlobalAveragePooling2D())
#model.add(layers.Dense(128, activation='relu', kernel_regularizer=regularizers.
#model.add(layers.Dropout(0.5))
#model.add(layers.Dense(NUM_CLASSES, activation='softmax'))

#lr_initial = 1e-4
#opt = optimizers.Adam(learning_rate=lr_initial, clipnorm=1.0)

#model.compile(
#    optimizer=opt,
#    loss='categorical_crossentropy',
#    metrics=['accuracy']
#)

#model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Para
input_layer (InputLayer)	(None, 224, 224, 3)	
Conv1 (Conv2D)	(None, 112, 112, 32)	
bn_Conv1 (BatchNormalization)	(None, 112, 112, 32)	
Conv1_relu (ReLU)	(None, 112, 112, 32)	
expanded_conv_depthwise (DepthwiseConv2D)	(None, 112, 112, 32)	
expanded_conv_depthwise_BN (BatchNormalization)	(None, 112, 112, 32)	
expanded_conv_depthwise_relu (ReLU)	(None, 112, 112, 32)	
expanded_conv_project (Conv2D)	(None, 112, 112, 16)	
expanded_conv_project_BN (BatchNormalization)	(None, 112, 112, 16)	
block_1_expand (Conv2D)	(None, 112, 112, 96)	1,
block_1_expand_BN (BatchNormalization)	(None, 112, 112, 96)	
block_1_expand_relu (ReLU)	(None, 112, 112, 96)	
block_1_pad (ZeroPadding2D)	(None, 113, 113, 96)	
block_1_depthwise (DepthwiseConv2D)	(None, 56, 56, 96)	
block_1_depthwise_BN (BatchNormalization)	(None, 56, 56, 96)	
block_1_depthwise_relu (ReLU)	(None, 56, 56, 96)	
block_1_project (Conv2D)	(None, 56, 56, 24)	2,
block_1_project_BN (BatchNormalization)	(None, 56, 56, 24)	
block_2_expand (Conv2D)	(None, 56, 56, 144)	3,
block_2_expand_BN (BatchNormalization)	(None, 56, 56, 144)	
block_2_expand_relu (ReLU)	(None, 56, 56, 144)	
block_2_depthwise (DepthwiseConv2D)	(None, 56, 56, 144)	1,
block_2_depthwise_BN (BatchNormalization)	(None, 56, 56, 144)	

block_2_depthwise_relu (ReLU)	(None, 56, 56, 144)	
block_2_project (Conv2D)	(None, 56, 56, 24)	3,
block_2_project_BN (BatchNormalization)	(None, 56, 56, 24)	
block_2_add (Add)	(None, 56, 56, 24)	
block_3_expand (Conv2D)	(None, 56, 56, 144)	3,
block_3_expand_BN (BatchNormalization)	(None, 56, 56, 144)	
block_3_expand_relu (ReLU)	(None, 56, 56, 144)	
block_3_pad (ZeroPadding2D)	(None, 57, 57, 144)	
block_3_depthwise (DepthwiseConv2D)	(None, 28, 28, 144)	1,
block_3_depthwise_BN (BatchNormalization)	(None, 28, 28, 144)	
block_3_depthwise_relu (ReLU)	(None, 28, 28, 144)	
block_3_project (Conv2D)	(None, 28, 28, 32)	4,
block_3_project_BN (BatchNormalization)	(None, 28, 28, 32)	
block_4_expand (Conv2D)	(None, 28, 28, 192)	6,
block_4_expand_BN (BatchNormalization)	(None, 28, 28, 192)	
block_4_expand_relu (ReLU)	(None, 28, 28, 192)	
block_4_depthwise (DepthwiseConv2D)	(None, 28, 28, 192)	1,
block_4_depthwise_BN (BatchNormalization)	(None, 28, 28, 192)	
block_4_depthwise_relu (ReLU)	(None, 28, 28, 192)	
block_4_project (Conv2D)	(None, 28, 28, 32)	6,
block_4_project_BN (BatchNormalization)	(None, 28, 28, 32)	
block_4_add (Add)	(None, 28, 28, 32)	
block_5_expand (Conv2D)	(None, 28, 28, 192)	6,
block_5_expand_BN (BatchNormalization)	(None, 28, 28, 192)	

block_5_expand_relu (ReLU)	(None, 28, 28, 192)	
block_5_depthwise (DepthwiseConv2D)	(None, 28, 28, 192)	1,
block_5_depthwise_BN (BatchNormalization)	(None, 28, 28, 192)	
block_5_depthwise_relu (ReLU)	(None, 28, 28, 192)	
block_5_project (Conv2D)	(None, 28, 28, 32)	6,
block_5_project_BN (BatchNormalization)	(None, 28, 28, 32)	
block_5_add (Add)	(None, 28, 28, 32)	
block_6_expand (Conv2D)	(None, 28, 28, 192)	6,
block_6_expand_BN (BatchNormalization)	(None, 28, 28, 192)	
block_6_expand_relu (ReLU)	(None, 28, 28, 192)	
block_6_pad (ZeroPadding2D)	(None, 29, 29, 192)	
block_6_depthwise (DepthwiseConv2D)	(None, 14, 14, 192)	1,
block_6_depthwise_BN (BatchNormalization)	(None, 14, 14, 192)	
block_6_depthwise_relu (ReLU)	(None, 14, 14, 192)	
block_6_project (Conv2D)	(None, 14, 14, 64)	12,
block_6_project_BN (BatchNormalization)	(None, 14, 14, 64)	
block_7_expand (Conv2D)	(None, 14, 14, 384)	24,
block_7_expand_BN (BatchNormalization)	(None, 14, 14, 384)	1,
block_7_expand_relu (ReLU)	(None, 14, 14, 384)	
block_7_depthwise (DepthwiseConv2D)	(None, 14, 14, 384)	3,
block_7_depthwise_BN (BatchNormalization)	(None, 14, 14, 384)	1,
block_7_depthwise_relu (ReLU)	(None, 14, 14, 384)	
block_7_project (Conv2D)	(None, 14, 14, 64)	24,
block_7_project_BN (BatchNormalization)	(None, 14, 14, 64)	

block_7_add (Add)	(None, 14, 14, 64)	
block_8_expand (Conv2D)	(None, 14, 14, 384)	24,
block_8_expand_BN (BatchNormalization)	(None, 14, 14, 384)	1,
block_8_expand_relu (ReLU)	(None, 14, 14, 384)	
block_8_depthwise (DepthwiseConv2D)	(None, 14, 14, 384)	3,
block_8_depthwise_BN (BatchNormalization)	(None, 14, 14, 384)	1,
block_8_depthwise_relu (ReLU)	(None, 14, 14, 384)	
block_8_project (Conv2D)	(None, 14, 14, 64)	24,
block_8_project_BN (BatchNormalization)	(None, 14, 14, 64)	
block_8_add (Add)	(None, 14, 14, 64)	
block_9_expand (Conv2D)	(None, 14, 14, 384)	24,
block_9_expand_BN (BatchNormalization)	(None, 14, 14, 384)	1,
block_9_expand_relu (ReLU)	(None, 14, 14, 384)	
block_9_depthwise (DepthwiseConv2D)	(None, 14, 14, 384)	3,
block_9_depthwise_BN (BatchNormalization)	(None, 14, 14, 384)	1,
block_9_depthwise_relu (ReLU)	(None, 14, 14, 384)	
block_9_project (Conv2D)	(None, 14, 14, 64)	24,
block_9_project_BN (BatchNormalization)	(None, 14, 14, 64)	
block_9_add (Add)	(None, 14, 14, 64)	
block_10_expand (Conv2D)	(None, 14, 14, 384)	24,
block_10_expand_BN (BatchNormalization)	(None, 14, 14, 384)	1,
block_10_expand_relu (ReLU)	(None, 14, 14, 384)	
block_10_depthwise (DepthwiseConv2D)	(None, 14, 14, 384)	3,

block_10_depthwise_BN (BatchNormalization)	(None, 14, 14, 384)	1,
block_10_depthwise_relu (ReLU)	(None, 14, 14, 384)	
block_10_project (Conv2D)	(None, 14, 14, 96)	36,
block_10_project_BN (BatchNormalization)	(None, 14, 14, 96)	
block_11_expand (Conv2D)	(None, 14, 14, 576)	55,
block_11_expand_BN (BatchNormalization)	(None, 14, 14, 576)	2,
block_11_expand_relu (ReLU)	(None, 14, 14, 576)	
block_11_depthwise (DepthwiseConv2D)	(None, 14, 14, 576)	5,
block_11_depthwise_BN (BatchNormalization)	(None, 14, 14, 576)	2,
block_11_depthwise_relu (ReLU)	(None, 14, 14, 576)	
block_11_project (Conv2D)	(None, 14, 14, 96)	55,
block_11_project_BN (BatchNormalization)	(None, 14, 14, 96)	
block_11_add (Add)	(None, 14, 14, 96)	
block_12_expand (Conv2D)	(None, 14, 14, 576)	55,
block_12_expand_BN (BatchNormalization)	(None, 14, 14, 576)	2,
block_12_expand_relu (ReLU)	(None, 14, 14, 576)	
block_12_depthwise (DepthwiseConv2D)	(None, 14, 14, 576)	5,
block_12_depthwise_BN (BatchNormalization)	(None, 14, 14, 576)	2,
block_12_depthwise_relu (ReLU)	(None, 14, 14, 576)	
block_12_project (Conv2D)	(None, 14, 14, 96)	55,
block_12_project_BN (BatchNormalization)	(None, 14, 14, 96)	
block_12_add (Add)	(None, 14, 14, 96)	
block_13_expand (Conv2D)	(None, 14, 14, 576)	55,

block_13_expand_BN (BatchNormalization)	(None, 14, 14, 576)	2,
block_13_expand_relu (ReLU)	(None, 14, 14, 576)	
block_13_pad (ZeroPadding2D)	(None, 15, 15, 576)	
block_13_depthwise (DepthwiseConv2D)	(None, 7, 7, 576)	5,
block_13_depthwise_BN (BatchNormalization)	(None, 7, 7, 576)	2,
block_13_depthwise_relu (ReLU)	(None, 7, 7, 576)	
block_13_project (Conv2D)	(None, 7, 7, 160)	92,
block_13_project_BN (BatchNormalization)	(None, 7, 7, 160)	
block_14_expand (Conv2D)	(None, 7, 7, 960)	153,
block_14_expand_BN (BatchNormalization)	(None, 7, 7, 960)	3,
block_14_expand_relu (ReLU)	(None, 7, 7, 960)	
block_14_depthwise (DepthwiseConv2D)	(None, 7, 7, 960)	8,
block_14_depthwise_BN (BatchNormalization)	(None, 7, 7, 960)	3,
block_14_depthwise_relu (ReLU)	(None, 7, 7, 960)	
block_14_project (Conv2D)	(None, 7, 7, 160)	153,
block_14_project_BN (BatchNormalization)	(None, 7, 7, 160)	
block_14_add (Add)	(None, 7, 7, 160)	
block_15_expand (Conv2D)	(None, 7, 7, 960)	153,
block_15_expand_BN (BatchNormalization)	(None, 7, 7, 960)	3,
block_15_expand_relu (ReLU)	(None, 7, 7, 960)	
block_15_depthwise (DepthwiseConv2D)	(None, 7, 7, 960)	8,
block_15_depthwise_BN (BatchNormalization)	(None, 7, 7, 960)	3,
block_15_depthwise_relu	(None, 7, 7, 960)	

(ReLU)		
block_15_project (Conv2D)	(None, 7, 7, 160)	153,
block_15_project_BN (BatchNormalization)	(None, 7, 7, 160)	
block_15_add (Add)	(None, 7, 7, 160)	
block_16_expand (Conv2D)	(None, 7, 7, 960)	153,
block_16_expand_BN (BatchNormalization)	(None, 7, 7, 960)	3,
block_16_expand_relu (ReLU)	(None, 7, 7, 960)	
block_16_depthwise (DepthwiseConv2D)	(None, 7, 7, 960)	8,
block_16_depthwise_BN (BatchNormalization)	(None, 7, 7, 960)	3,
block_16_depthwise_relu (ReLU)	(None, 7, 7, 960)	
block_16_project (Conv2D)	(None, 7, 7, 320)	307,
block_16_project_BN (BatchNormalization)	(None, 7, 7, 320)	1,
Conv_1 (Conv2D)	(None, 7, 7, 1280)	409,
Conv_1_bn (BatchNormalization)	(None, 7, 7, 1280)	5,
out_relu (ReLU)	(None, 7, 7, 1280)	
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	
dropout (Dropout)	(None, 1280)	
dense (Dense)	(None, 7)	8,

Total params: 2,266,951 (8.65 MB)

Trainable params: 728,967 (2.78 MB)

Non-trainable params: 1,537,984 (5.87 MB)

Agora iremos realizar o balanceamento das classes e executar o fit do modelo

In [23]:
```
#-------------------------------------------------------------------------
# Treinando parando o modelo CNN
#-------------------------------------------------------------------------

classes = list(train_gen.class_indices.keys())
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)

# ---------------------------------------------------------------------
# Calculando peso das classes
# ---------------------------------------------------------------------
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(train_gen.classes),
    y=train_gen.classes
)

# ---------------------------------------------------------------------
# Convertendo pesos para dicionário do KERAS
# ---------------------------------------------------------------------
class_weights = dict(enumerate(class_weights))
print("\nPesos de classe calculados:", class_weights)

history = model.fit(
    train_gen,
    epochs=20,
    validation_data=val_gen,
    class_weight=class_weights,
    callbacks=[
        tf.keras.callbacks.ModelCheckpoint(
            'tech_challenge.keras',
            monitor='val_accuracy',
            save_best_only=True
        ),
        tf.keras.callbacks.EarlyStopping(
            monitor='val_accuracy',
            patience=7,
            restore_best_weights=True,
            mode='max'
        ),
        tf.keras.callbacks.ReduceLROnPlateau(
            monitor='val_accuracy',
            factor=0.5,
            patience=3,
            min_lr=1e-6
        )
    ],
    verbose=1
)
```

```
Distribuição das classes no conjunto de validação:

Carcinoma basocelular            64
Dermatofibroma                   15
Lesão benigna tipo queratose    135
Lesão vascular                   17
Melanoma                        139
Nevo melanocítico               829
Queratose actínica               41
Name: count, dtype: int64

Pesos de classe calculados: {0: np.float64(2.7606679035250465), 1: np.float64(12.
358803986710964), 2: np.float64(1.3073273589878756), 3: np.float64(10.21978021978
0219), 4: np.float64(1.2728828058169375), 5: np.float64(0.21376853235260315), 6:
np.float64(4.338192419825073)}
Epoch 1/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 77s 316ms/step - accuracy: 0.3615 - loss: 1.7345 - v
al_accuracy: 0.4879 - val_loss: 1.5943 - learning_rate: 1.0000e-04
Epoch 2/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 76s 318ms/step - accuracy: 0.5232 - loss: 1.2355 - v
al_accuracy: 0.5887 - val_loss: 1.2869 - learning_rate: 1.0000e-04
Epoch 3/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 74s 312ms/step - accuracy: 0.5742 - loss: 1.0928 - v
al_accuracy: 0.5427 - val_loss: 1.4894 - learning_rate: 1.0000e-04
Epoch 4/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 74s 310ms/step - accuracy: 0.5867 - loss: 1.0691 - v
al_accuracy: 0.5879 - val_loss: 1.3160 - learning_rate: 1.0000e-04
Epoch 5/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 74s 309ms/step - accuracy: 0.6085 - loss: 0.9175 - v
al_accuracy: 0.6395 - val_loss: 1.1122 - learning_rate: 1.0000e-04
Epoch 6/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 73s 308ms/step - accuracy: 0.6367 - loss: 0.8737 - v
al_accuracy: 0.5984 - val_loss: 1.2619 - learning_rate: 1.0000e-04
Epoch 7/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 76s 321ms/step - accuracy: 0.6415 - loss: 0.8656 - v
al_accuracy: 0.6806 - val_loss: 1.0272 - learning_rate: 1.0000e-04
Epoch 8/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 76s 321ms/step - accuracy: 0.6558 - loss: 0.8011 - v
al_accuracy: 0.7169 - val_loss: 0.9514 - learning_rate: 1.0000e-04
Epoch 9/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 76s 319ms/step - accuracy: 0.6585 - loss: 0.7784 - v
al_accuracy: 0.6548 - val_loss: 1.0243 - learning_rate: 1.0000e-04
Epoch 10/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 75s 317ms/step - accuracy: 0.6646 - loss: 0.7339 - v
al_accuracy: 0.7024 - val_loss: 0.9340 - learning_rate: 1.0000e-04
Epoch 11/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 76s 318ms/step - accuracy: 0.6922 - loss: 0.6989 - v
al_accuracy: 0.6492 - val_loss: 1.0584 - learning_rate: 1.0000e-04
Epoch 12/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 76s 318ms/step - accuracy: 0.6972 - loss: 0.6438 - v
al_accuracy: 0.6137 - val_loss: 1.1084 - learning_rate: 5.0000e-05
Epoch 13/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 75s 317ms/step - accuracy: 0.6938 - loss: 0.6394 - v
al_accuracy: 0.6032 - val_loss: 1.1424 - learning_rate: 5.0000e-05
Epoch 14/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 76s 318ms/step - accuracy: 0.7065 - loss: 0.6085 - v
al_accuracy: 0.7145 - val_loss: 0.8456 - learning_rate: 5.0000e-05
Epoch 15/20
233/233 ━━━━━━━━━━━━━━━━━━━━ 76s 317ms/step - accuracy: 0.7216 - loss: 0.5888 - v
al_accuracy: 0.6839 - val_loss: 0.8975 - learning_rate: 2.5000e-05
```

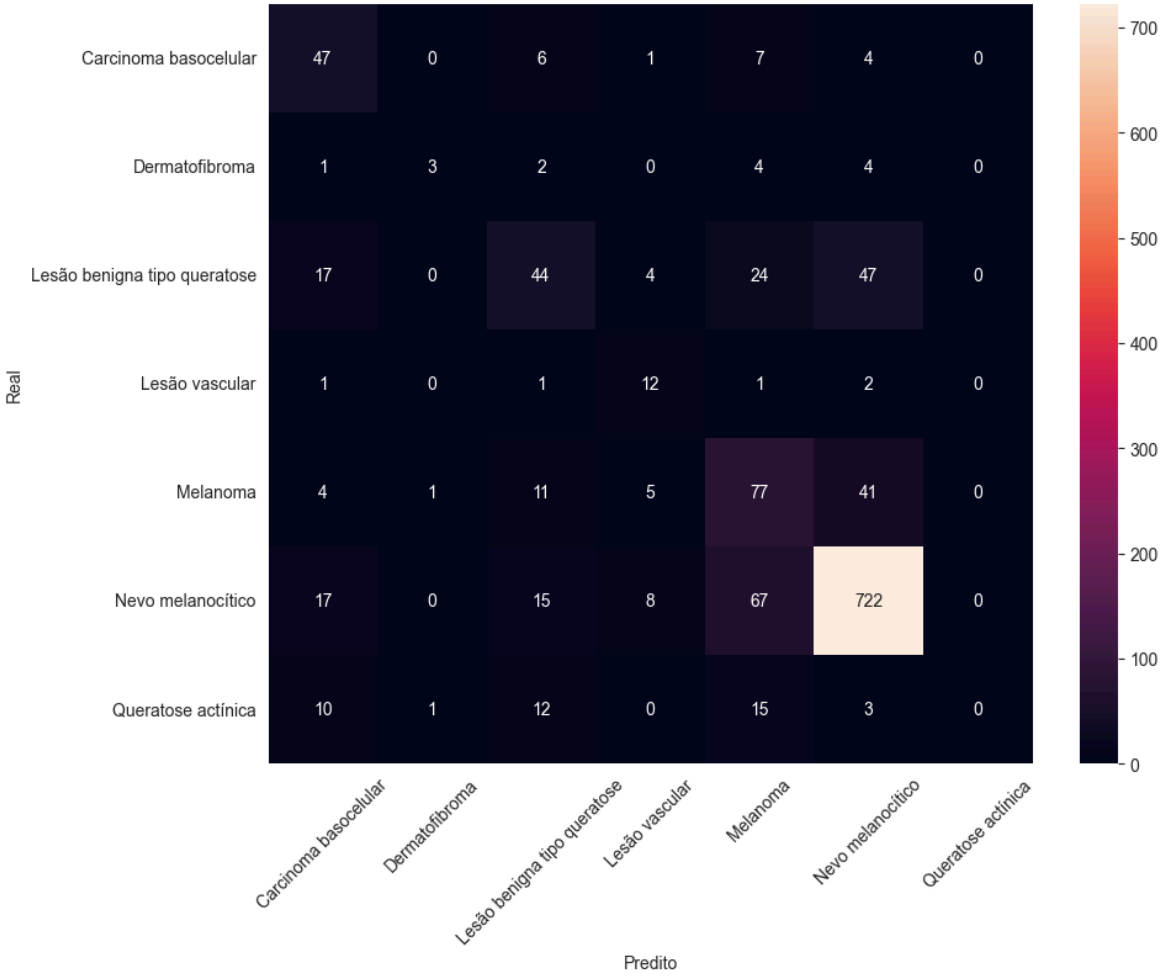
🤔 4. Avaliação

Abaixo iremos avaliar o modelo treinado e a precisão alcançada para cada uma das classes treinadas do campo "diagnóstico"

In [24]:
```
#--------------------------------------------------------------------------
# Avaliação
#--------------------------------------------------------------------------
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
))
```

```
39/39 ━━━━━━━━━━━━━━━━━━━━ 7s 177ms/step
                           precision    recall  f1-score   support

       Carcinoma basocelular       0.48      0.72      0.58        65
              Dermatofibroma       0.60      0.21      0.32        14
Lesão benigna tipo queratose       0.48      0.32      0.39       136
             Lesão vascular       0.40      0.71      0.51        17
                   Melanoma       0.39      0.55      0.46       139
           Nevo melanocítico       0.88      0.87      0.87       829
          Queratose actínica       0.00      0.00      0.00        41

                    accuracy                           0.73      1241
                   macro avg       0.46      0.48      0.45      1241
                weighted avg       0.72      0.73      0.72      1241
```

In [25]:
```
#--------------------------------------------------------------------------
# Matriz de confusão
#--------------------------------------------------------------------------
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()
```



Agora iremos gerar imagens por meio do GRAND CAM para analisar os padrões de imagem treinados por nosso modelo CNN.

In [26]:
```
# ------------------------------------------------------------------
# Extraindo Mapa de Grad-CAM
# ------------------------------------------------------------------
def encontrar_ultima_conv(model):
    ultima_conv = None

    def procurar_camadas(m):
        nonlocal ultima_conv
        for layer in m.layers:
            if hasattr(layer, 'layers') and len(layer.layers) > 0:
                procurar_camadas(layer)
            if isinstance(layer, (tf.keras.layers.Conv2D,
                                  tf.keras.layers.DepthwiseConv2D,
                                  tf.keras.layers.SeparableConv2D)):
                ultima_conv = layer.name

    procurar_camadas(model)
    return ultima_conv

ultima_conv = encontrar_ultima_conv(model)
print("Última camada convolucional encontrada:", 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
```

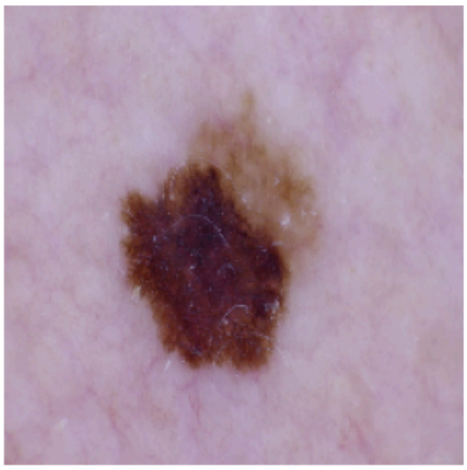
In [27]:

```
# ------------------------------------------------------------------
# Exibindo Grad-CAM em lote
# ------------------------------------------------------------------
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=(8, num_imagens * 3))

    for i, row in exemplos.iterrows():
        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

        mapa = gerar_mapa_gradcam(model, entrada, ultima_conv)
        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
```

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

        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=15)
```

Imagem Original: Nevo melanocítico



Mapa Grad-CAM

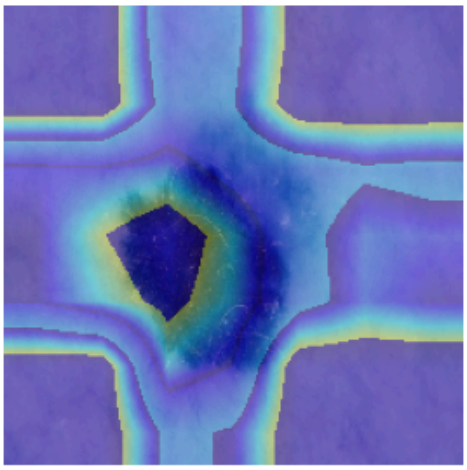
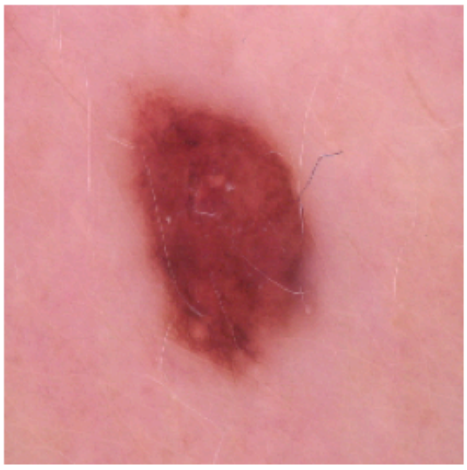


Imagem Original: Nevo melanocítico



Mapa Grad-CAM

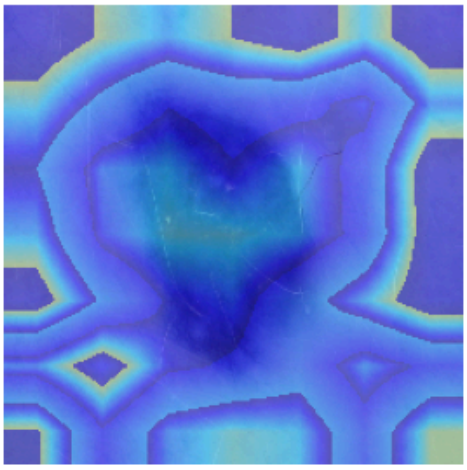
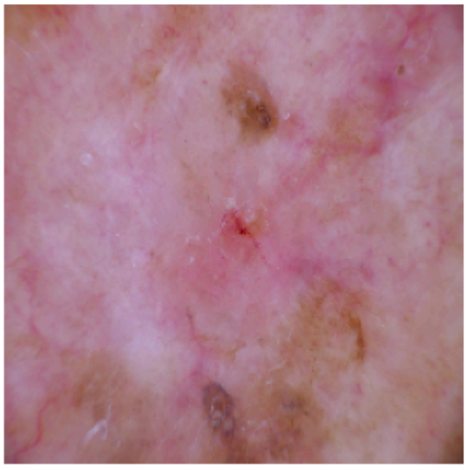


Imagem Original: Carcinoma basocelular



Mapa Grad-CAM

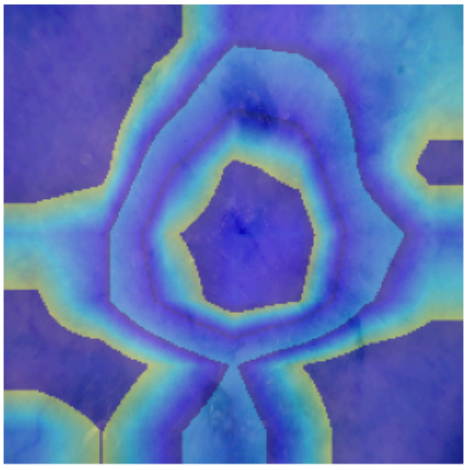
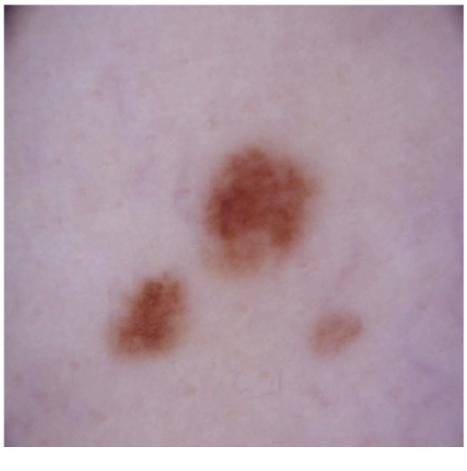


Imagem Original: Nevo melanocítico



Mapa Grad-CAM

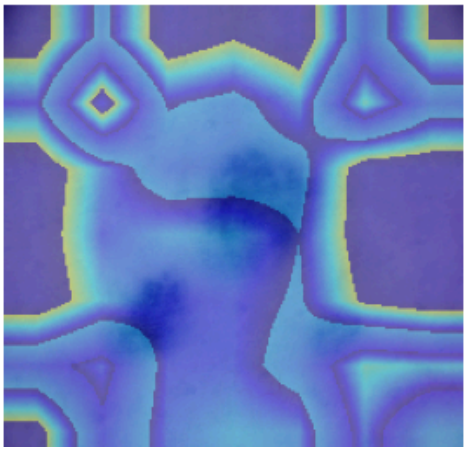
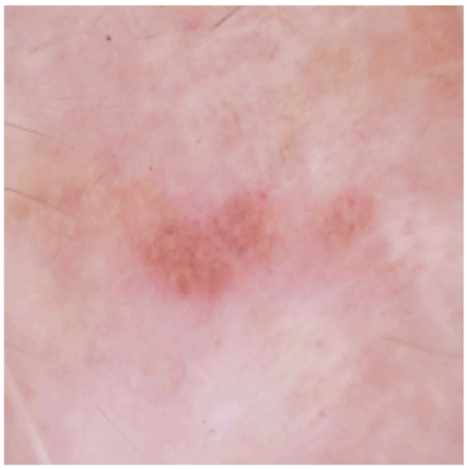


Imagem Original: Carcinoma basocelular



Mapa Grad-CAM

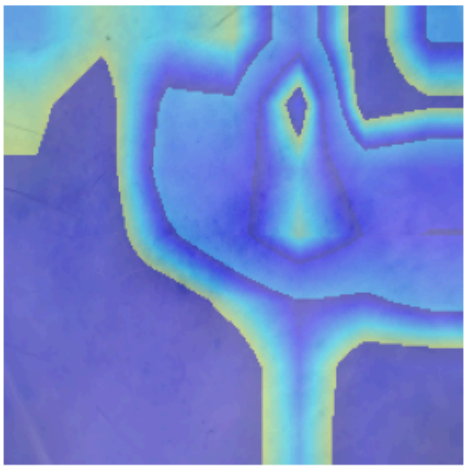
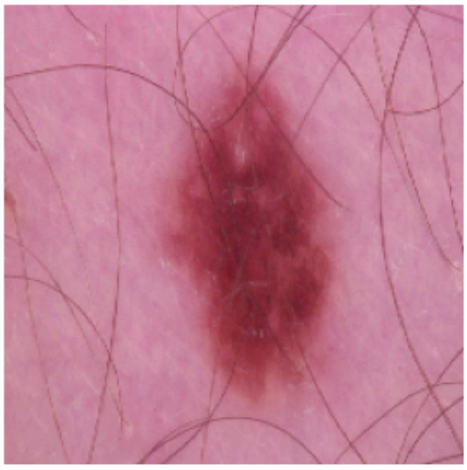


Imagem Original: Nevo melanocítico



Mapa Grad-CAM

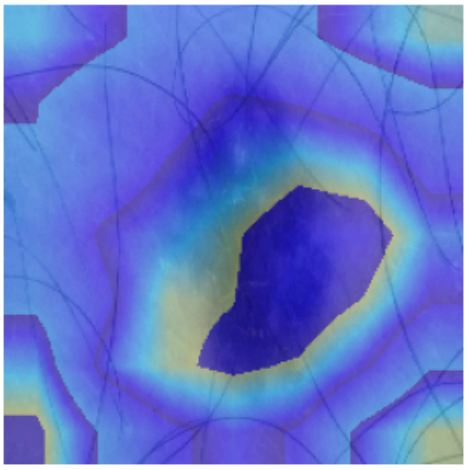


Imagem Original: Nevo melanocítico

Mapa Grad-CAM

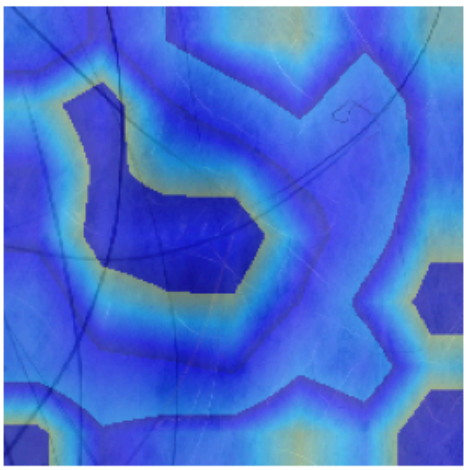
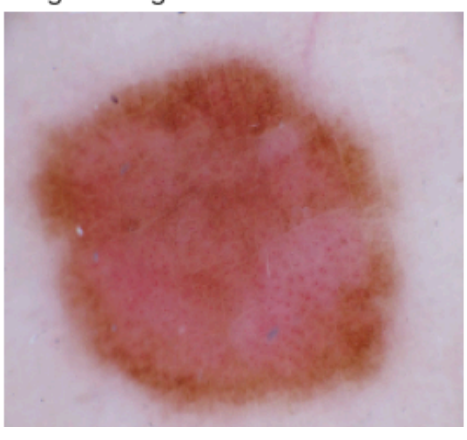


Imagem Original: Nevo melanocítico



Mapa Grad-CAM

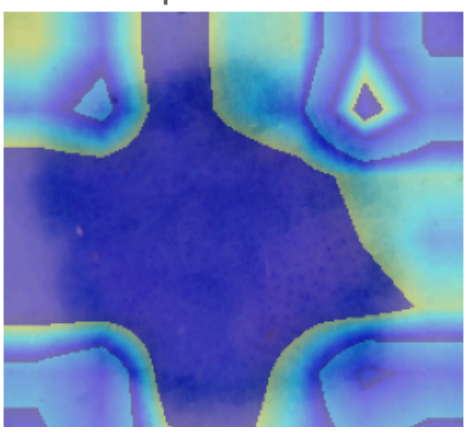
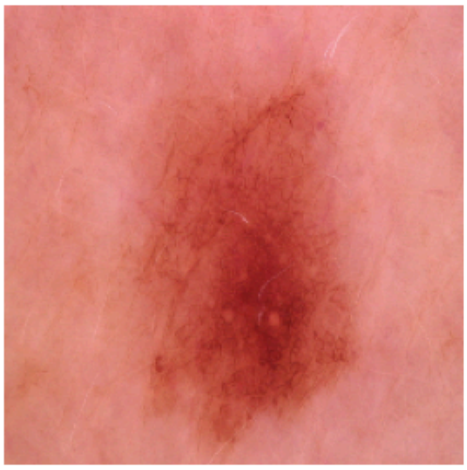


Imagem Original: Nevo melanocítico

Mapa Grad-CAM



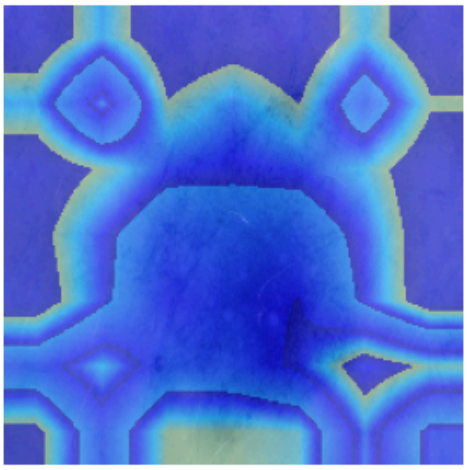
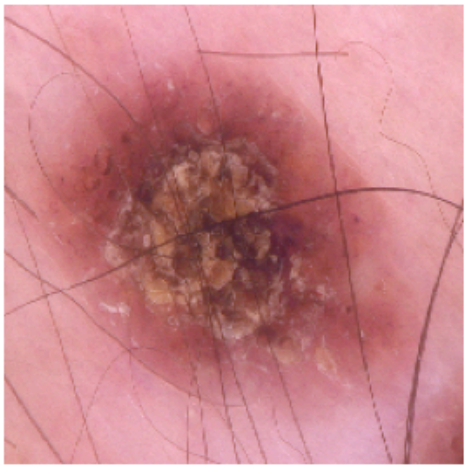


Imagem Original: Lesão benigna tipo queratose

Mapa Grad-CAM



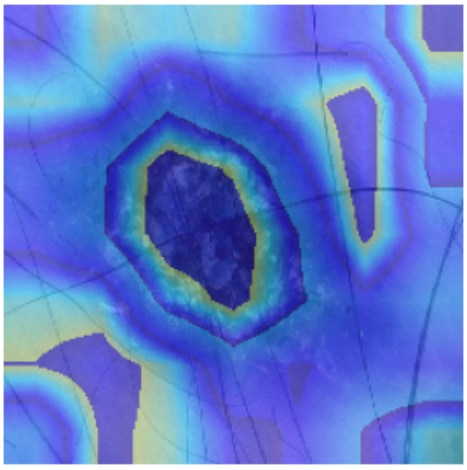
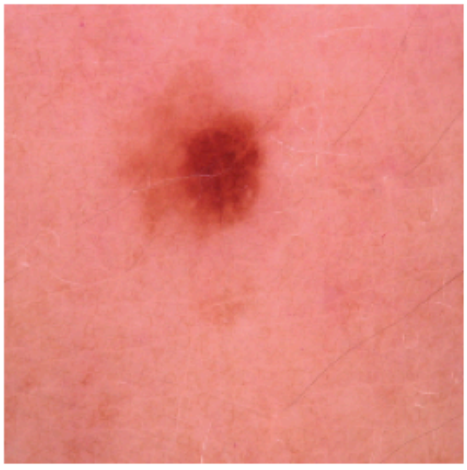


Imagem Original: Nevo melanocítico

Mapa Grad-CAM



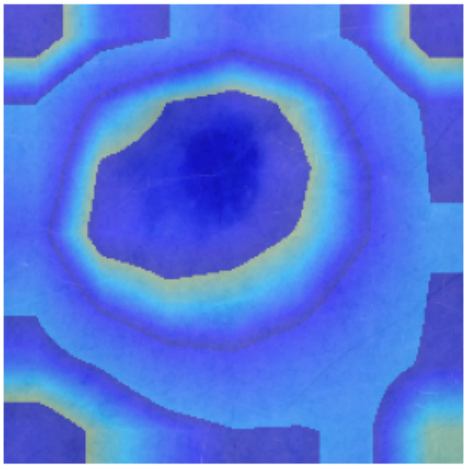
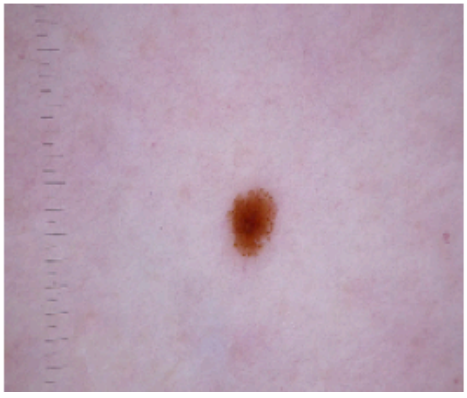


Imagem Original: Nevo melanocítico

Mapa Grad-CAM



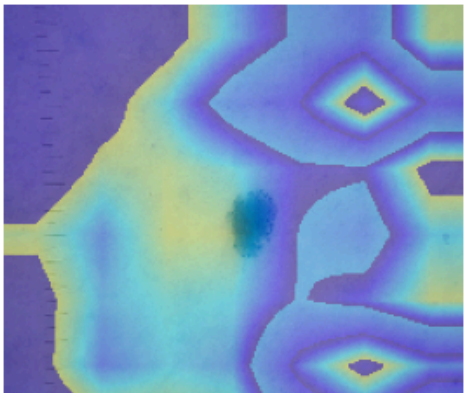
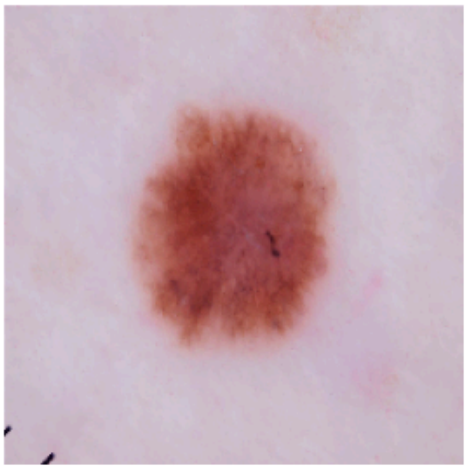


Imagem Original: Nevo melanocítico

Mapa Grad-CAM



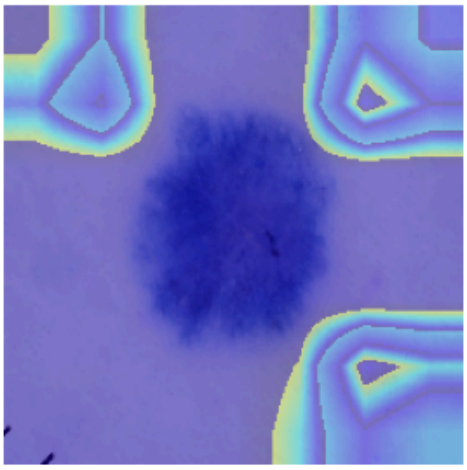


Imagem Original: Nevo melanocítico

Mapa Grad-CAM

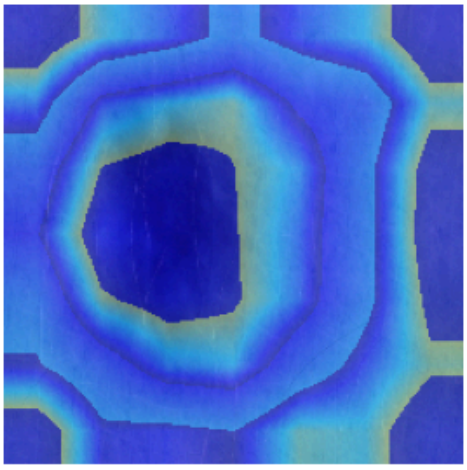
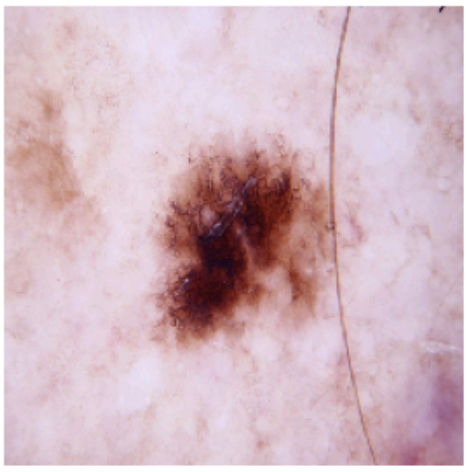
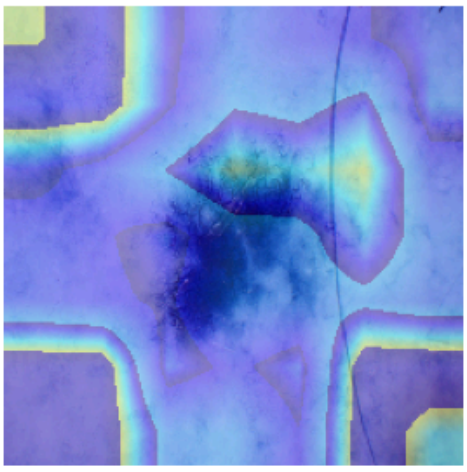


Imagem Original: Lesão benigna tipo queratose

Mapa Grad-CAM





💻 5. Utilizando o Modelo

Agora iremos utilizar o nosso modelo treinado para validar imagens de um dataset apenas de imagens e fazer o mesmo a identificar o tipo de lesão e se possui câncer ou não.

In [29]:
```
#-------------------------------------------------------------------------
# Utilização do modelo para Predição
#-------------------------------------------------------------------------
```

```
rows = []
for root, dirs, files in os.walk(dataset_file2):
    for f in files:
        if f.lower().endswith(('.jpg', '.jpeg', '.png')):
            full = os.path.join(root, f)
            parent = Path(root).name.lower()
            rows.append({'image_path': full, 'parent_folder': parent})

df_isic = pd.DataFrame(rows)
print(f"Total imagens encontradas: {len(df_isic)}")
display(df_isic.head())

if df_isic.empty:
    raise RuntimeError("Nenhuma imagem encontrada, por favor verifique o caminho

NUM_SAMPLES = min(30, len(df_isic))
sample_df = df_isic.sample(NUM_SAMPLES, random_state=42).reset_index(drop=True)
print(f"Gerando um total de {NUM_SAMPLES} imagens para predição.")

class_indices = getattr(train_gen, "class_indices", None)
if class_indices is None:
    raise RuntimeError("Train_Gen não encontrado.")

inv_class_indices = {v: k for k, v in class_indices.items()}
print("Mapeamento de índices ->:")
print(inv_class_indices)

def preprocess_image_for_model(img_path, target_size):
    img = load_img(img_path, target_size=target_size)
    img_arr = img_to_array(img)
    if img_arr.shape[-1] == 4:
        img_arr = img_arr[..., :3]
    if img_arr.ndim == 2:
        img_arr = np.stack([img_arr]*3, axis=-1)
    img_arr = img_arr / 255.0
    return img_arr

X = []
paths = []
for p in sample_df['image_path']:
    try:
        arr = preprocess_image_for_model(p, IMG_SIZE)
    except Exception as e:
        print("Erro ao carregar:", p, e)
        continue
    X.append(arr)
    paths.append(p)

if len(X) == 0:
    raise RuntimeError("Nenhuma imagem válida para a predição.")

X = np.array(X, dtype=np.float32)

#------------------------------------------------------------------------------
# Predição
#------------------------------------------------------------------------------
pred_probs = model.predict(X, verbose=0)
pred_idx = np.argmax(pred_probs, axis=1)
pred_class_names = [inv_class_indices[i] for i in pred_idx]
```

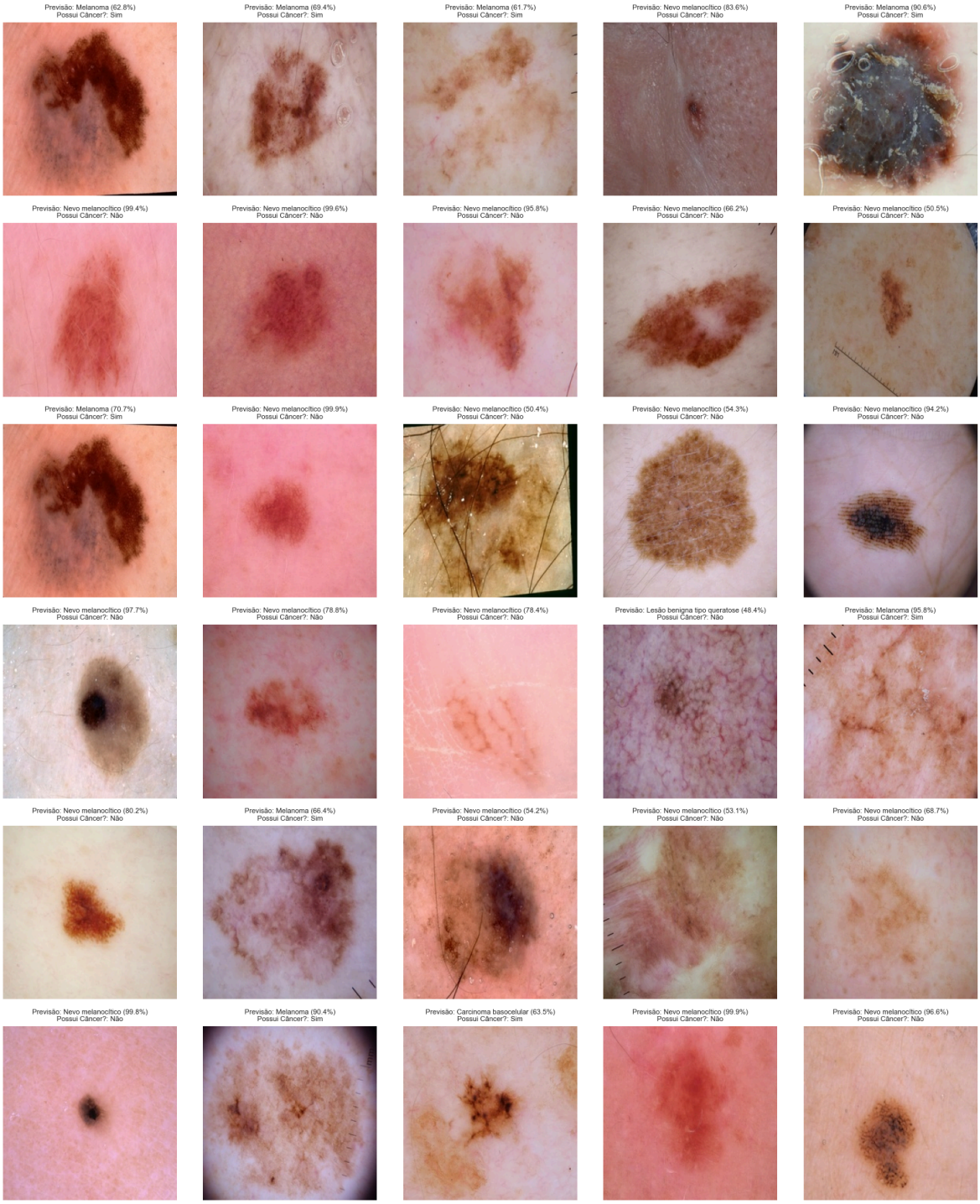
```
pred_confidences = pred_probs[np.arange(len(pred_idx)), pred_idx]

#--------------------------------------------------------------------------
# Identificar se é câncer a partir de pred_class
#--------------------------------------------------------------------------
def is_cancer_by_pred_class(nome):
    if nome in ["Melanoma", "Carcinoma basocelular", "Queratose actínica"]:
        return 1
    return 0

pred_cancer = [is_cancer_by_pred_class(n) for n in pred_class_names]

results_df = pd.DataFrame({
    'image_path': paths,
    'pred_idx': pred_idx,
    'pred_class': pred_class_names,
    'pred_confidence': pred_confidences,
    'pred_cancer': pred_cancer
})

display(results_df.head(10))

#--------------------------------------------------------------------------
# Exibição de amostras (sem GT)
#--------------------------------------------------------------------------
n = len(results_df)
cols = 5
rows = max(1, (n + cols - 1) // cols)
plt.figure(figsize=(cols*3.2, rows*3.2))

for i, row in results_df.reset_index().iterrows():
    ax = plt.subplot(rows, cols, i+1)
    img = load_img(row['image_path'], target_size=IMG_SIZE)
    ax.imshow(img)

    pred_text = f"Previsão: {row['pred_class']} ({row['pred_confidence']*100:.1f
    cancer_text = 'Possui Câncer?: Sim' if row['pred_cancer']==1 else 'Possui Câ

    ax.set_title(f"{pred_text}\n{cancer_text}", fontsize=8)
    ax.axis('off')

plt.tight_layout()
plt.show()

#--------------------------------------------------------------------------
# Salvar resultados
#--------------------------------------------------------------------------
results_df.to_csv('resultado_modelo.csv', index=False)
print("Resultados salvos em resultado_modelo.csv")
```

```
Total imagens encontradas: 3297
```

	image_path	parent_folder
0	C:\Users\flmli\.cache\kagglehub\datasets\rm100...	benign
1	C:\Users\flmli\.cache\kagglehub\datasets\rm100...	benign
2	C:\Users\flmli\.cache\kagglehub\datasets\rm100...	benign
3	C:\Users\flmli\.cache\kagglehub\datasets\rm100...	benign
4	C:\Users\flmli\.cache\kagglehub\datasets\rm100...	benign

```
Gerando um total de 30 imagens para predição.
Mapeamento de índices ->:
{0: 'Carcinoma basocelular', 1: 'Dermatofibroma', 2: 'Lesão benigna tipo queratos
e', 3: 'Lesão vascular', 4: 'Melanoma', 5: 'Nevo melanocítico', 6: 'Queratose act
ínica'}
```

	image_path	pred_idx	pred_class	pred_confidence
0	C:\Users\flmli\.cache\kagglehub\datasets\rm100...	4	Melanoma	0.627967
1	C:\Users\flmli\.cache\kagglehub\datasets\rm100...	4	Melanoma	0.694017
2	C:\Users\flmli\.cache\kagglehub\datasets\rm100...	4	Melanoma	0.617027
3	C:\Users\flmli\.cache\kagglehub\datasets\rm100...	5	Nevo melanocítico	0.836006
4	C:\Users\flmli\.cache\kagglehub\datasets\rm100...	4	Melanoma	0.906399
5	C:\Users\flmli\.cache\kagglehub\datasets\rm100...	5	Nevo melanocítico	0.993893
6	C:\Users\flmli\.cache\kagglehub\datasets\rm100...	5	Nevo melanocítico	0.996084
7	C:\Users\flmli\.cache\kagglehub\datasets\rm100...	5	Nevo melanocítico	0.957711
8	C:\Users\flmli\.cache\kagglehub\datasets\rm100...	5	Nevo melanocítico	0.662039
9	C:\Users\flmli\.cache\kagglehub\datasets\rm100...	5	Nevo melanocítico	0.505415



Resultados salvos em resultado_modelo.csv