CURSO: <BÁSICO EM MACHINE LEARNING>

- **Atividade 05 (ATIV-05)**
  - Tipo: Somativa.
  - Tema: Resumo sobre pré-processamento de imagens, segmentação de imagens e detecção/classificação de imagens.
  - Conteúdo: Módulo 3.
  - Participantes: individual.
  - Avaliação do aluno.
    - Objetivo: Avaliar desempenho do aluno sobre técnicas de classificação/segmentação de imagens.
    - Nota: 0 a 3 supercrítico, 4 a 6 crítico, 5 a 7 razoável e 8 a 10 bom.
    - Critérios avaliados: Coerência, coesão no texto e organização das informações. Também são avaliados bibliotecas/frameworks e exemplos de aplicações com códigos comentados.
  - Informações complementares: Atividade alternativa para substituir a atividade em grupo ATIV-04.
  - **AO CONCLUIR A ATIVIDADE: ENVIAR APENAS O LINK DO REPOSITÓRIO GITHUB (ESPECIFICAR A BRANCH) PÚBLICO COM O PDF DA ATIVIDADE.**

ITENS:

1. Pré-processamento de imagens;
2. Segmentação de imagens;
3. Detecção/classificação de imagens;

Você deve realizar um resumo (no máximo duas páginas para cada item) para os item

INFORMAÇÕES ESPERADAS no resumo de cada item:

- Introdução;
- Exemplos de bibliotecas/frameworks;
- Exemplos de aplicações (mostre códigos simples e explique);

O pré-processamento de imagens é uma etapa crucial para melhorar a qualidade visual e garantir que análises subsequentes, como a segmentação, sejam precisas. Durante a aquisição de imagens, podem surgir diversos defeitos que precisam ser corrigidos para minimizar imperfeições e destacar detalhes essenciais. Essa fase é fundamental para assegurar que a análise dos dados produz resultados confiáveis.

Existem várias técnicas de pré-processamento que podem ser aplicadas, dependendo do tipo de defeito na imagem. Por exemplo, melhorias no brilho e contraste são úteis para destacar detalhes em imagens subexpostas ou superexpostas. A redução de ruídos pode ser necessária quando a imagem contém interferências indesejadas, enquanto a correção de iluminação irregular ajuda a uniformizar a luminosidade em toda a imagem. Outras técnicas, como o realce de bordas, são usadas para tornar as transições entre diferentes regiões mais nítidas.

Em Python, bibliotecas como OpenCV, Scikit-image , Pillow e Matplotlib são amplamente utilizadas para implementar essas técnicas, elas são instaladas usando o pip install, por exemplo pip install OpenCV-python. Algumas das operações mais comuns incluem:

- Redimensionar o tamanho: Alterar o tamanho de uma imagem para um formato específico, frequentemente necessário para assegurar a consistência dos dados em modelos de aprendizado de máquina ou para diminuir o tempo de processamento. É utilizado o comando resized_image = cv2.resize(image, (new_width, new_height)).
- Conversão para escala de cinza: É comum converter imagens coloridas para escala de cinza para simplificar o processamento. Com a biblioteca OpenCV, essa conversão é feita com o comando gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY).
- Redução de ruído: Técnicas como o filtro Gaussiano (cv2.GaussianBlur) são utilizadas para suavizar a imagem e reduzir ruídos, facilitando a detecção de características importantes.
- Correção de perspectiva: Se uma imagem estiver distorcida devido ao ângulo de captura, a correção de perspectiva pode ser aplicada usando funções como cv2.getPerspectiveTransform e cv2.warpPerspective para alinhar corretamente a imagem.
- Detecção de bordas: Métodos como o detector de bordas de Canny (cv2.Canny) são usados para identificar contornos e formas dentro da imagem, o que é especialmente útil em aplicações de reconhecimento de objetos.

Isso ilustra a variedade de ferramentas disponíveis para o pré-processamento de imagens, cada uma desempenhando um papel específico na preparação dos dados para análises mais profundas.

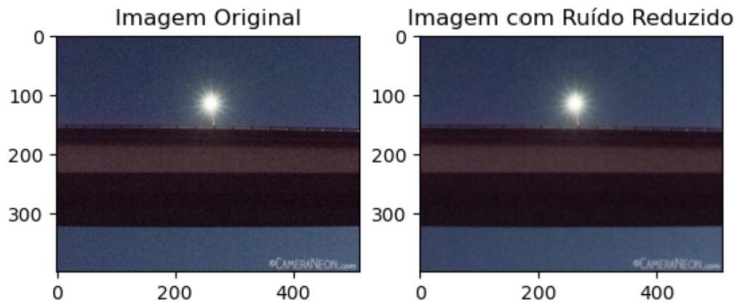
Aqui está um exemplo da técnica de filtro Gaussiano:

```python
import cv2
import matplotlib.pyplot as plt

# Carrega a imagem
image = cv2.imread('ruido.jpg')

# Aplica o filtro Gaussiano para reduzir o ruído
blurred_image = cv2.GaussianBlur(image, (5, 5), 0) # (5, 5) é o tamanho do kernel (núcleo) do filtro

# Exibe a imagem original e a imagem com ruído reduzido
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Imagem Original')
plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(blurred_image, cv2.COLOR_BGR2RGB))
plt.title('Imagem com Ruído Reduzido')
plt.show()
```



Este código utiliza a biblioteca OpenCV para aplicar um filtro Gaussiano em uma imagem, suavizando-a e reduzindo o ruído. Ele primeiro carrega a imagem e, em seguida, aplica a função cv2.GaussianBlur para suavizar a imagem usando um kernel Gaussiano com tamanho definido por (5, 5). O resultado é uma imagem com ruído reduzido. Por fim, o código exibe a imagem original e a imagem suavizada lado a lado usando a biblioteca Matplotlib, para que os resultados possam ser comparados.

Quando falamos de pré-processamento de imagens, depois dessa etapa, o próximo passo é a segmentação. Basicamente, isso significa dividir a imagem em partes menores ou objetos. Cada uma dessas partes é uniforme e homogênea em relação a algumas características da imagem, como cor e textura.

As técnicas tradicionais de segmentação de imagem utilizam uma variedade de bibliotecas e frameworks para implementar e executar algoritmos de processamento de imagem e aprendizado de máquina. Aqui estão algumas das bibliotecas e frameworks mais utilizados para cada uma das técnicas tradicionais mencionadas:

1. Limite (Thresholding)
   - OpenCV: A função cv2.threshold() pode ser usada para aplicar limiarização básica, enquanto a função cv2.adaptiveThreshold() oferece métodos mais sofisticados. O método de Otsu pode ser facilmente aplicado usando cv2.threshold() com o tipo cv2.THRESH_OTSU.
   - Scikit-image: A função skimage.filters.threshold_otsu() implementa o método de Otsu.
2. Segmentação Baseada na Região (Region-based Segmentation)
   - Scikit-image: A função skimage.segmentation.flood_fill() pode ser usada para crescimento de regiões. Para segmentação por regiões,

skimage.segmentation.quickshift() e skimage.segmentation.slic() são métodos comumente usados.
    - OpenCV: Embora menos comum para crescimento de regiões, o OpenCV pode implementar esse método com lógica customizada combinada com limiarização e operações morfológicas.
3. Segmentação Baseada em Agrupamento (Clustering)
    - Scikit-learn: Esta biblioteca é amplamente utilizada para aprendizado de máquina, incluindo o algoritmo K-means, com a função sklearn.cluster.KMeans.
    - OpenCV: Implementa K-means usando cv2.kmeans(), que pode ser utilizado diretamente em imagens para segmentação baseada em agrupamento.

As Redes Neurais Convolucionais (CNNs) entram em cena principalmente em métodos modernos de segmentação de imagem, especialmente quando se busca uma precisão maior ou quando o problema exige a captura de informações contextuais complexas. Enquanto as técnicas tradicionais de segmentação de imagem, como as que mencionei anteriormente, dependem de características simples como cor, brilho, contraste, e bordas, as CNNs podem aprender automaticamente características de alto nível diretamente dos dados de imagem.

Para implementar CNNs para segmentação de imagem, os seguintes frameworks são amplamente utilizados:

1. TensorFlow
    - TensorFlow permite a segmentação semântica de imagens, onde cada pixel é classificado em uma categoria específica, como objetos, pessoas ou áreas, essencial para tarefas em medicina, como a análise de imagens médicas, e em visão computacional.
    - O framework facilita a implementação de redes neurais convolucionais (CNNs) avançadas, como U-Net e Mask R-CNN, que são amplamente utilizadas para segmentação de imagens, oferecendo alta precisão e capacidade de identificar tanto classes de objetos quanto suas instâncias individuais.
    - Pode-se customizar modelos de segmentação de imagem, ajustando hiperparâmetros e arquiteturas para atender a requisitos específicos do projeto, desde segmentação simples até tarefas complexas que exigem modelos personalizados.
    - TensorFlow aproveita GPUs e TPUs para acelerar o treinamento de modelos de segmentação, permitindo o processamento eficiente de grandes volumes de dados de imagem, o que é crucial para aplicações que requerem rápida iteração e ajuste fino.
    - Usando TensorBoard, pode-se visualizar a performance dos modelos de segmentação em tempo real, acompanhando métricas como a precisão da segmentação, erros de classificação de pixels, e evolução do treinamento, facilitando o refinamento do modelo.
2. PyTorch
    - PyTorch é amplamente utilizado para segmentação de imagem, suportando arquiteturas avançadas como U-Net e Mask R-CNN, que permitem a classificação precisa de pixels e a identificação de instâncias individuais em uma imagem.

- É oferecido uma estrutura flexível para construir e ajustar modelos de segmentação, facilitando a personalização e a experimentação com novas arquiteturas e parâmetros.
- Otimizado para rodar em GPUs, PyTorch permite treinar modelos de segmentação de forma rápida e eficiente, essencial para o processamento de grandes conjuntos de dados de imagem.

Aqui está um exemplo de código contendo técnica de segmentação Thresholding:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Carrega a imagem
image = cv2.imread('download (1).jpg')

# Converte a imagem para o espaço de cores HSV (Hue, Saturation, Value)
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# Define os intervalos de cores para a segmentação
lower_color = np.array([20, 50, 50])  # Limite inferior da cor no espaço HSV
upper_color = np.array([40, 255, 255])  # Limite superior da cor no espaço HSV

# Cria uma máscara usando os intervalos de cores definidos
color_mask = cv2.inRange(hsv_image, lower_color, upper_color)

# Aplica a máscara à imagem original
segmented_image = cv2.bitwise_and(image, image, mask=color_mask)

plt.subplot(1, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Imagem Original')
plt.subplot(1, 3, 2)
plt.imshow(hsv_image, cmap='gray')
plt.title('hsv image')
plt.subplot(1, 3, 3)
plt.imshow(segmented_image, cmap='gray')
plt.title('segmented_image')
plt.show()
```

Este código usa a biblioteca OpenCV para segmentar uma imagem com base em sua cor. Ele primeiro carrega a imagem e a converte para o espaço de cores HSV, que é mais adequado para segmentação. Então, define limites inferior e superior para a faixa de cores desejada no espaço HSV e cria uma máscara que identifica pixels com essas cores. A máscara é aplicada à imagem original usando a operação "AND" bit a bit, resultando em uma imagem segmentada que contém apenas os pixels dentro do intervalo de cor especificado. Finalmente, o código exibe a imagem original, a imagem no espaço HSV e a imagem segmentada lado a lado usando a biblioteca Matplotlib.

Depois do pré-processamento e da segmentação, ocorre a classificação. As redes neurais artificiais são essenciais para o deep learning. Inspiradas no funcionamento do cérebro humano, essas redes consistem em camadas de neurônios artificiais interligados. No contexto da classificação de imagens, elas são treinadas para identificar padrões e características específicas, permitindo uma categorização precisa.

Os frameworks mais utilizados para classificação de imagens em deep learning incluem:

1. TensorFlow/Keras
    - Utiliza-se o Keras para construir um modelo de rede neural convolucional (CNN), responsável por extrair características relevantes das imagens para a classificação. A CNN pode ser criada do zero com camadas convolucionais (`Conv2D`), de pooling (`MaxPooling2D`) e densas (`Dense`), ou pode-se optar por utilizar uma arquitetura pré-treinada como VGG, ResNet ou MobileNet. Após

definir a arquitetura, o modelo é compilado com um otimizador adequado, uma função de perda específica, e métricas como a precisão.

- O modelo é treinado utilizando os dados de treinamento, onde parâmetros como a taxa de aprendizado e o número de épocas são ajustados para maximizar o desempenho. Durante o treinamento, monitoram-se continuamente a perda e a precisão, permitindo ajustes finos no modelo conforme necessário para evitar problemas como overfitting.
- Após o treinamento, o modelo é avaliado em um conjunto de dados de teste para verificar sua capacidade de generalização. Métricas como precisão, recall e F1-score são calculadas para quantificar a eficácia do modelo em classificar corretamente as imagens em suas respectivas classes, oferecendo uma visão clara de seu desempenho real.

2. OpenCV: Embora mais focado em processamento de imagens, pode ser integrado com TensorFlow para criar pipelines completos de classificação de imagens.

Aqui está um exemplo de classificação de imagem utilizando Keras:

```python
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
import matplotlib.pyplot as plt

# Carregar o dataset CIFAR-10
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

# Normalizar os dados
train_images, test_images = train_images / 255.0, test_images / 255.0

# Construir o modelo
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# Compilar o modelo
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Treinar o modelo
model.fit(train_images, train_labels, epochs=10, validation_split=0.2)

# Avaliar o modelo
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Acurácia no conjunto de teste: {test_acc}')

# Fazer previsões
predictions = model.predict(test_images)

# Mostrar a primeira imagem de teste e a sua previsão
plt.imshow(test_images[0])
plt.title(f'Previsão: {tf.argmax(predictions[0])}')
plt.show()
```

O código em TensorFlow/Keras executa a tarefa de classificação de imagens utilizando uma rede neural convolucional (CNN). Primeiramente, o dataset CIFAR-10 é carregado, composto por 60.000 imagens coloridas divididas em 10 categorias, como aviões, carros e pássaros. As imagens passam por um processo de normalização, ajustando os valores dos pixels para o

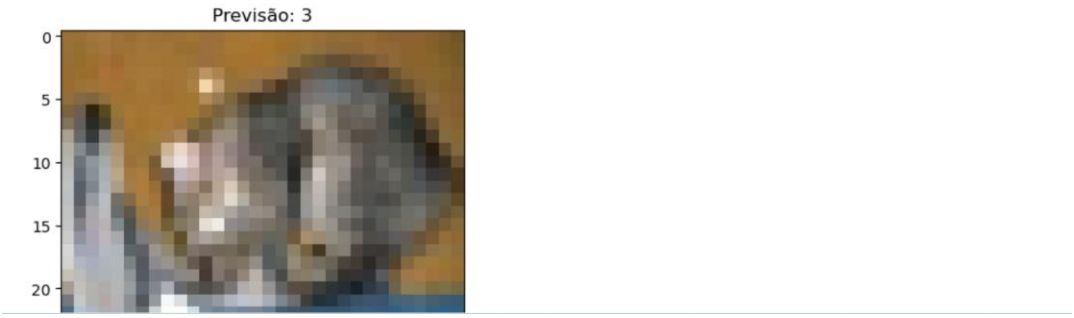
intervalo de [0, 1], o que facilita a convergência do modelo durante o treinamento. A CNN é construída com várias camadas, incluindo camadas convolucionais para a extração de características e camadas de pooling para a redução da dimensionalidade. A camada final utiliza a função de ativação softmax para calcular a probabilidade de a imagem pertencer a uma das 10 classes.

O modelo é então compilado e treinado ao longo de 10 épocas, utilizando 80% das imagens para o treinamento e reservando 20% para a validação. Após o treinamento, o desempenho do modelo é avaliado com o conjunto de dados de teste, verificando sua capacidade de generalização. Esse processo permite ao modelo aprender a identificar padrões específicos nas imagens que correspondem a diferentes categorias, como animais ou objetos, e avaliar sua precisão na classificação dessas categorias.

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 64s 0us/step

C:\Users\paula\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `inp
ut_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Epoch 1/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 32s 19ms/step - accuracy: 0.3258 - loss: 1.8141 - val_accuracy: 0.5199 - val_loss: 1.3368
Epoch 2/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 23s 19ms/step - accuracy: 0.5454 - loss: 1.2666 - val_accuracy: 0.5838 - val_loss: 1.1841
Epoch 3/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 20s 16ms/step - accuracy: 0.6185 - loss: 1.0826 - val_accuracy: 0.6452 - val_loss: 1.0157
Epoch 4/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - accuracy: 0.6605 - loss: 0.9599 - val_accuracy: 0.6590 - val_loss: 0.9844
Epoch 5/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - accuracy: 0.6899 - loss: 0.8764 - val_accuracy: 0.6775 - val_loss: 0.9328
Epoch 6/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 42s 18ms/step - accuracy: 0.7162 - loss: 0.8095 - val_accuracy: 0.6846 - val_loss: 0.9205
Epoch 7/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 42s 18ms/step - accuracy: 0.7383 - loss: 0.7512 - val_accuracy: 0.6962 - val_loss: 0.8927
Epoch 8/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 24s 19ms/step - accuracy: 0.7502 - loss: 0.7100 - val_accuracy: 0.6796 - val_loss: 0.9449
Epoch 9/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 40s 18ms/step - accuracy: 0.7689 - loss: 0.6536 - val_accuracy: 0.7017 - val_loss: 0.8890
Epoch 10/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 24s 19ms/step - accuracy: 0.7878 - loss: 0.6060 - val_accuracy: 0.7018 - val_loss: 0.8990
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.6944 - loss: 0.9079
Acurácia no conjunto de teste: 0.6919000148773193
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step
```



No final, ao realizar a classificação de uma imagem de teste, o modelo retornou o valor "3", indicando que a imagem foi classificada como pertencente à classe com o índice 3 do CIFAR-10, que corresponde à categoria "cat" (gato). Isso reflete a capacidade do modelo em identificar a imagem como sendo de um gato, baseado nos padrões aprendidos durante o treinamento.