PROCESSAMENTO DE IMAGENS

Classificação de imagens

Docente: Felipe Viel
Discente: Enzo Luigi Schork e Vinicius Borchardt Grisa

Inspeção de Edificações no Setor da Engenharia Civil

Os Desafios da Abordagem Tradicional

- Tempo e Custo Elevados
- Subjetividade e Falhas Humanas
- Dificuldade de Rastreabilidade

Os Desafios da Abordagem por meio de Visão Computacional

- Escopo fechado
- Alto custo computacional
- Dificuldade de interpretabilidade

Realizações propostas

- Manipulação dos dados
- Análise dos dados por meio de redes neurais convolucionais (CNNs)
- Classificação dos resultados
- Análise dos resultados

Benefícios da Solução

- Agilidade
- Precisão
- Escalabilidade
- Maior Rastreabilidade

Pipeline de desenvolvimento

1 - Particionar dados
2 - Divisão dos dados em conjuntos
3 - Aplicação de ruído
4 - Pré-processamento
5 - Construir arquitetura do modelo
6 - Treinamento do modelo
7 - Predição de novas imagens individuais
8 - Aplicação de métricas para análise

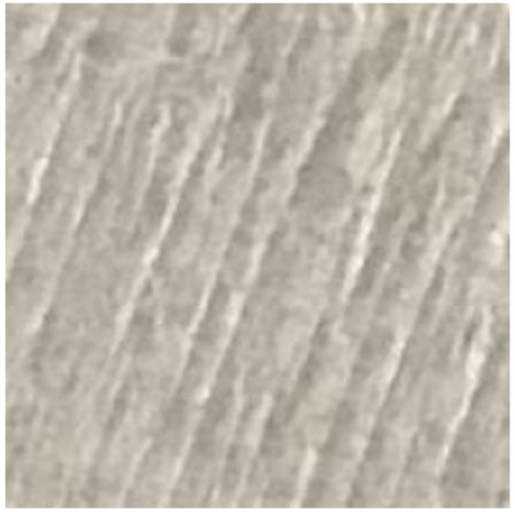
1 e 2 - Particionar e dividir em conjuntos

```python
DATA_DIR = os.path.join(EXTRACT_DIR, 'meu_dataset_concreto')

# definições iniciais
IMG_HEIGHT = 227
IMG_WIDTH = 227
BATCH_SIZE = 32


# carrega os dados pós upload
# treino
train_ds = tf.keras.utils.image_dataset_from_directory(
    directory=os.path.join(DATA_DIR, 'train'),
    labels='inferred',
    label_mode='int',
    image_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=42
)

# validação
val_ds = tf.keras.utils.image_dataset_from_directory(
    directory=os.path.join(DATA_DIR, 'validation'),
    labels='inferred',
    label_mode='int',
    image_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    shuffle=False
)

# teste
test_ds = tf.keras.utils.image_dataset_from_directory(
    directory=os.path.join(DATA_DIR, 'test'),
    labels='inferred',
    label_mode='int',
    image_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    shuffle=False
)
```

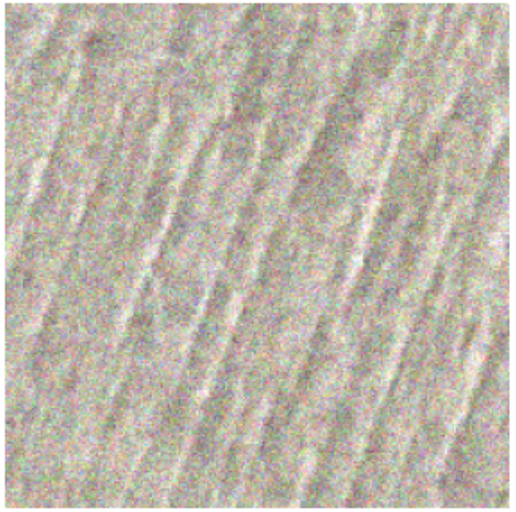
3 - Aplicação de ruído

```python
# adição de ruído
def add_gaussian_noise(image, mean=0, var=0.01):
    sigma = var**0.5
    gauss = np.random.normal(mean, sigma, image.shape)
    noisy_image = image + gauss
    noisy_image = np.clip(noisy_image, 0, 1)
    return noisy_image.astype(np.float32)

print("\nAdding Gaussian noise to the dataset...")
X_train_noisy = np.array([add_gaussian_noise(img) for img in X_train])
X_val_noisy = np.array([add_gaussian_noise(img) for img in X_val])
X_test_noisy = np.array([add_gaussian_noise(img) for img in X_test])
print("Noise added to training, validation, and test sets.")
```

Original Sample



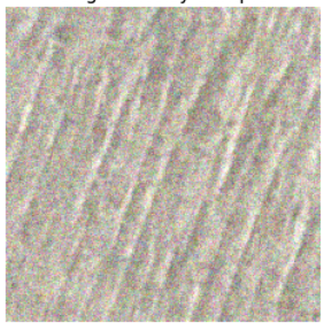
Noisy Sample (Gaussian)



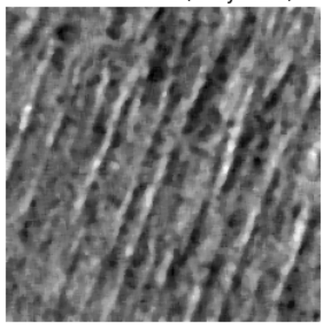
4 - Pré processamento

```python
def apply_image_processing_flow(images_array):
    processed_images = []
    for img in images_array:
        img_uint8 = (img * 255).astype(np.uint8)

        # aplicar grayscale
        if len(img_uint8.shape) == 3 and img_uint8.shape[2] == 3:
            img_gray = cv2.cvtColor(img_uint8, cv2.COLOR_RGB2GRAY)
        else:
            img_gray = img_uint8

        # filtro de mediana
        filtered_img_gray = cv2.medianBlur(img_gray, 5) # 5x5 kernel

        # equalização de histograma
        equalized_img_gray = cv2.equalizeHist(filtered_img_gray)

        processed_images.append(equalized_img_gray)

    processed_images = np.array(processed_images)
    if len(processed_images.shape) == 3:
        processed_images = np.expand_dims(processed_images, axis=-1)

    return processed_images.astype('float32') / 255.0
```

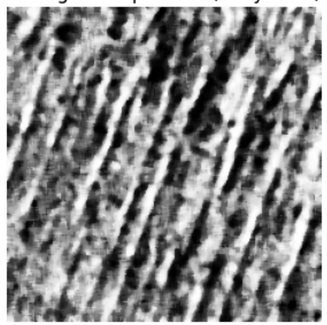
Original Noisy Sample



Median Filtered (Grayscale)



Histogram Equalized (Grayscale)



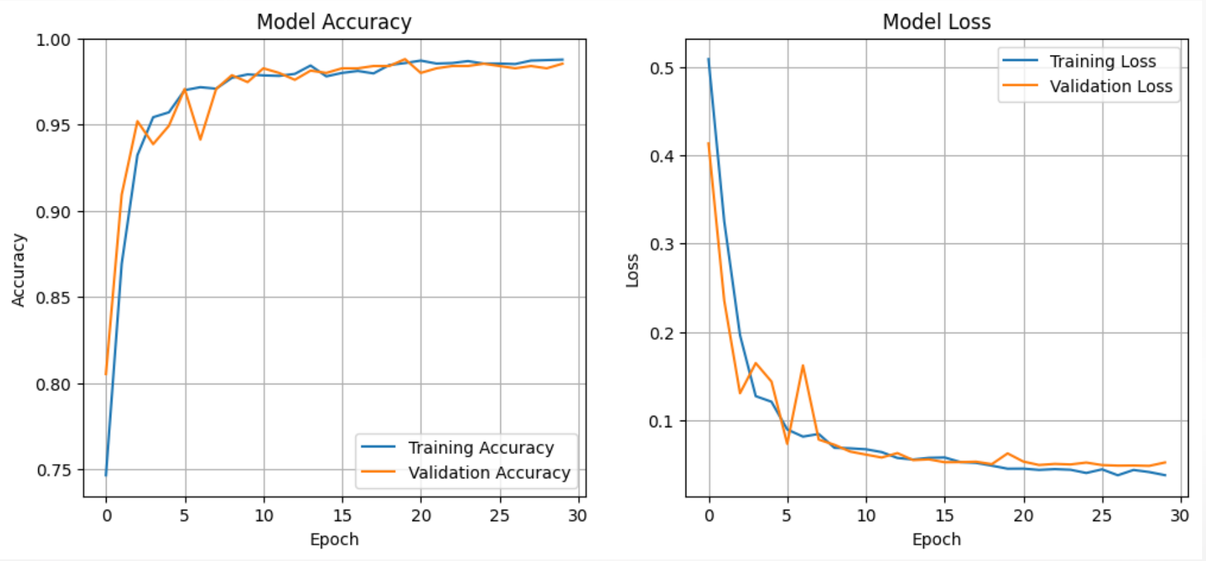
5 - Arquitetura do modelo

```python
# definir modelo
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    GlobalAveragePooling2D(),  # substitui flatten
    Dense(64, activation='relu'),  # reduzido de 128 para 64
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])

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

print("\n--- Initial CNN Model Architecture ---")
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 225, 225, 32)	320
max_pooling2d (MaxPooling2D)	(None, 112, 112, 32)	0
conv2d_1 (Conv2D)	(None, 110, 110, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 55, 55, 64)	0
conv2d_2 (Conv2D)	(None, 53, 53, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 128)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 128)	0
dense (Dense)	(None, 64)	8,256
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 2)	130

6 - Treinamento do modelo

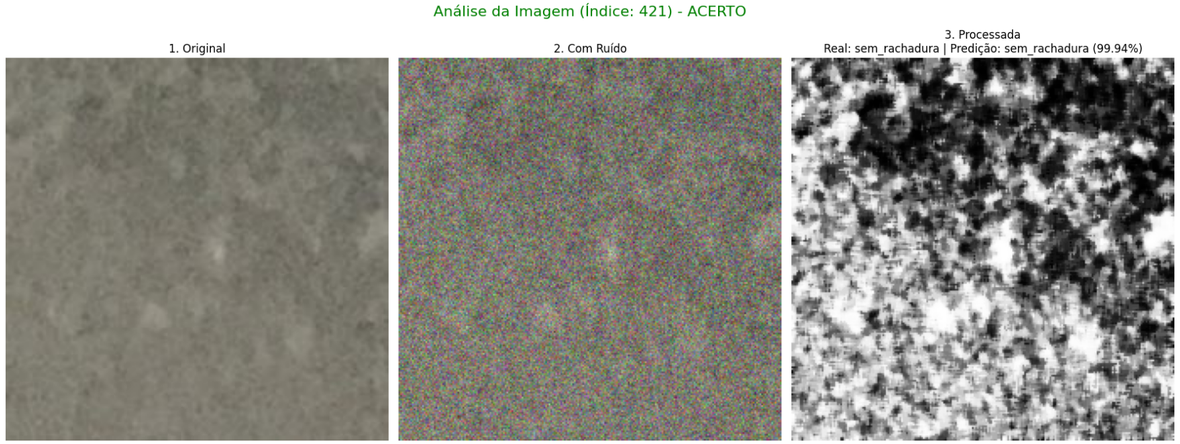
```python
# callbacks para evitar overfitting e ajustar lr
callbacks = [
    EarlyStopping(monitor='val_loss', patience=4, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2)
]

# treinar modelo
print("\n--- Training Initial CNN Model ---")
history = model.fit(
    X_train_processed, y_train,
    epochs=30, # aumentar épocas
    validation_data=(X_val_processed, y_val),
    batch_size=32,
    callbacks=callbacks
)
```



```
--- Training Initial CNN Model ---
Epoch 1/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 18s 94ms/step - accuracy: 0.6005 - loss: 0.6389 - val_accuracy: 0.8133 - val_loss: 0.4795 - learning_rate: 0.0010
Epoch 2/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 10s 45ms/step - accuracy: 0.8669 - loss: 0.3402 - val_accuracy: 0.9293 - val_loss: 0.1790 - learning_rate: 0.0010
Epoch 3/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 44ms/step - accuracy: 0.9387 - loss: 0.1650 - val_accuracy: 0.9347 - val_loss: 0.1555 - learning_rate: 0.0010
Epoch 4/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - accuracy: 0.9564 - loss: 0.1315 - val_accuracy: 0.9787 - val_loss: 0.0689 - learning_rate: 0.0010
Epoch 5/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 44ms/step - accuracy: 0.9718 - loss: 0.0813 - val_accuracy: 0.9747 - val_loss: 0.0670 - learning_rate: 0.0010
Epoch 6/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 44ms/step - accuracy: 0.9648 - loss: 0.0923 - val_accuracy: 0.9773 - val_loss: 0.0871 - learning_rate: 0.0010
Epoch 7/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 44ms/step - accuracy: 0.9684 - loss: 0.0913 - val_accuracy: 0.9693 - val_loss: 0.0747 - learning_rate: 0.0010
Epoch 8/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 44ms/step - accuracy: 0.9751 - loss: 0.0728 - val_accuracy: 0.9813 - val_loss: 0.0576 - learning_rate: 5.0000e-04
Epoch 9/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 45ms/step - accuracy: 0.9748 - loss: 0.0651 - val_accuracy: 0.9813 - val_loss: 0.0550 - learning_rate: 5.0000e-04
Epoch 10/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 44ms/step - accuracy: 0.9803 - loss: 0.0599 - val_accuracy: 0.9813 - val_loss: 0.0602 - learning_rate: 5.0000e-04
Epoch 11/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 48ms/step - accuracy: 0.9791 - loss: 0.0602 - val_accuracy: 0.9827 - val_loss: 0.0525 - learning_rate: 5.0000e-04
Epoch 12/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 10s 45ms/step - accuracy: 0.9823 - loss: 0.0523 - val_accuracy: 0.9853 - val_loss: 0.0510 - learning_rate: 5.0000e-04
Epoch 13/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 45ms/step - accuracy: 0.9797 - loss: 0.0557 - val_accuracy: 0.9840 - val_loss: 0.0508 - learning_rate: 5.0000e-04
Epoch 14/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 45ms/step - accuracy: 0.9808 - loss: 0.0547 - val_accuracy: 0.9787 - val_loss: 0.0669 - learning_rate: 5.0000e-04
Epoch 15/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 45ms/step - accuracy: 0.9857 - loss: 0.0492 - val_accuracy: 0.9813 - val_loss: 0.0539 - learning_rate: 5.0000e-04
Epoch 16/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 45ms/step - accuracy: 0.9811 - loss: 0.0576 - val_accuracy: 0.9853 - val_loss: 0.0502 - learning_rate: 2.5000e-04
Epoch 17/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 48ms/step - accuracy: 0.9850 - loss: 0.0459 - val_accuracy: 0.9853 - val_loss: 0.0475 - learning_rate: 2.5000e-04
Epoch 18/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 45ms/step - accuracy: 0.9857 - loss: 0.0426 - val_accuracy: 0.9867 - val_loss: 0.0457 - learning_rate: 2.5000e-04
Epoch 19/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 44ms/step - accuracy: 0.9842 - loss: 0.0523 - val_accuracy: 0.9867 - val_loss: 0.0493 - learning_rate: 2.5000e-04
Epoch 20/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 45ms/step - accuracy: 0.9851 - loss: 0.0521 - val_accuracy: 0.9813 - val_loss: 0.0594 - learning_rate: 2.5000e-04
Epoch 21/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 44ms/step - accuracy: 0.9802 - loss: 0.0502 - val_accuracy: 0.9867 - val_loss: 0.0465 - learning_rate: 1.2500e-04
Epoch 22/30
110/110 ━━━━━━━━━━━━━━━━━━━━ 5s 45ms/step - accuracy: 0.9880 - loss: 0.0390 - val_accuracy: 0.9840 - val_loss: 0.0480 - learning_rate: 1.2500e-04
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.
```

7 - Predição de novas imagens

```
print("Carregando modelo treinado do arquivo...")
loaded_model = load_model('modelo_rachaduras_final.h5')
print("Modelo carregado com sucesso.")

# seleciona imagem aleatoria do dataset de teste
random_index = np.random.randint(0, len(X_test))

# pré processamento
original_image = X_test[random_index]
noisy_image = X_test_noisy[random_index]
processed_image = X_test_processed[random_index]
true_label_index = y_test[random_index]
true_label_name = class_names[true_label_index]

image_for_prediction = np.expand_dims(processed_image, axis=0)


# predição
prediction = loaded_model.predict(image_for_prediction) # <-- Usando o 'loaded_model'
predicted_label_index = np.argmax(prediction[0])
predicted_label_name = class_names[predicted_label_index]
confidence = np.max(prediction[0])
```



8 - Aplicação de métricas de análise

```python
# carregar modelo
loaded_model = load_model('modelo_rachaduras_final.h5')

# obter probabilidades de predição
y_pred_probs = loaded_model.predict(X_test_processed)

# probabilidades -> classe
y_pred = np.argmax(y_pred_probs, axis=1)

print("Predições geradas com sucesso.")
print("\n--- Relatório de Classificação ---")
print(classification_report(y_test, y_pred, target_names=class_names))

# calcula matriz
cm = confusion_matrix(y_test, y_pred)

# resultados
plt.figure(figsize=(8, 6))
sns.heatmap(cm,
            annot=True,
            fmt='d',
            cmap='Blues',
            xticklabels=class_names,
            yticklabels=class_names)

plt.ylabel('Real (Verdadeiro)')
plt.xlabel('Predição (Previsto)')
plt.title('Matriz de Confusão', fontsize=16)
plt.show()
```

```
--- Relatório de Classificação ---
              precision    recall  f1-score   support

com_rachadura       0.99      1.00      1.00       375
sem_rachadura       1.00      0.99      1.00       375

     accuracy                           1.00       750
    macro avg       1.00      1.00      1.00       750
 weighted avg       1.00      1.00      1.00       750
```