INTELIGÊNCIA COMPUTACIONAL

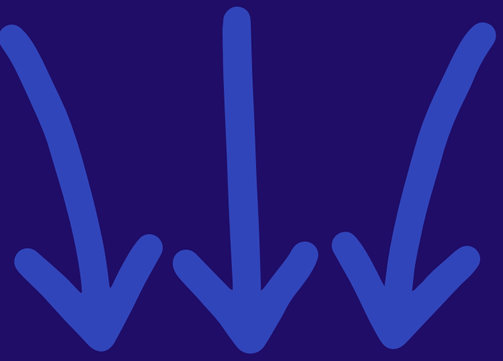
CLASSIFICAÇÃO DE MATERIAL CIRURGICO

Inês Ferreira a2023111442
Tiago Manata a2023147352

setembro, 2025

DESCRIÇÃO E OBJETIVOS DO PROBLEMA

Erros no bloco custam vidas



- **Qualquer troca ou demora na entrega de instrumentos gera risco cirúrgico. O enfermeiro instrumentista decide em segundos.**
- **Risco de contaminação do ambiente estéril do bloco operatório.**

A medicina deve acompanhar o avanço das novas tecnologias.

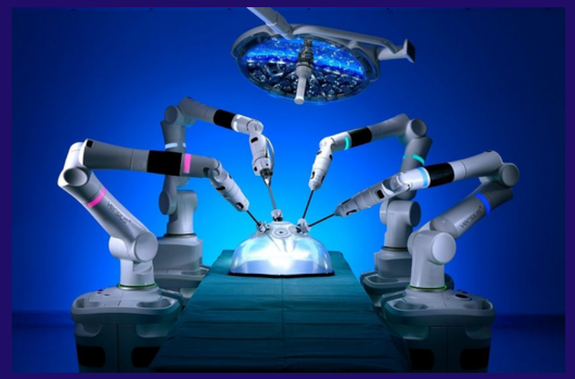
A IA pode reduzir o risco, minimizar erros e tornar os cuidados mais seguros.

DESCRIÇÃO E OBJETIVOS DO PROBLEMA

Objetivos

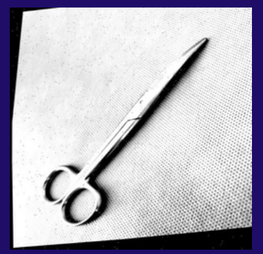
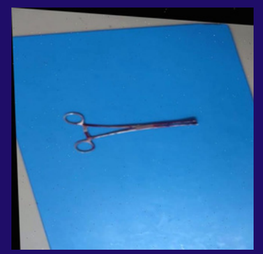
Implementar uma rede neuronal

Identificar automaticamente material cirúrgico



DATASET E TRATAMENTO DE IMAGEM

CLASSES

Bisturi	Tesoura de Mayo Curva	Pinça Hemostática	Pinça	Tesoura de Mayo Reta
				

Recolha Manual de fotos

Kaggle

Repositório Mendeley

Roboflow

Data Augmentation

DATASET E TRATAMENTO DE IMAGEM

MLP

```python
def load_images(root, size=(32, 32)):
    X, y = [], []
    for folder in sorted(p for p in root.iterdir() if p.is_dir()):
        for file in folder.glob("*"):
            try:
                img = Image.open(file).convert("L").resize(size)
                X.append(np.asarray(img, dtype=np.float32).ravel())
                y.append(folder.name)
            except:
                continue
    return np.array(X), np.array(y)
```

CNN

```python
def load_images(root, size=(32, 32)):
    X, y = [], []
    for folder in sorted(p for p in root.iterdir() if p.is_dir()):
        for file in folder.glob("*"):
            img = Image.open(file).convert("RGB").resize(size)
            X.append(np.asarray(img, dtype=np.float32) / 255.0)
            y.append(folder.name)
    return np.array(X), np.array(y)

X, y_text = load_images(DATA_DIR)
```

MLP e CNN

```python
X, y_text = load_images(DATA_DIR)

le = LabelEncoder()
y = le.fit_transform(y_text)
```

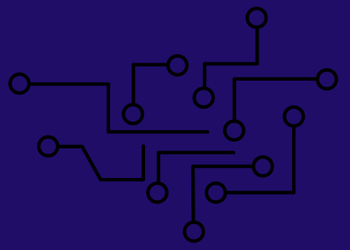
Transforma os nomes das classes em números

CNN

```python
from tensorflow.keras.utils import to_categorical
yTrain_binario = to_categorical(y_train, 5)
yTest_binario = to_categorical(y_test, 5)
```

One-hot encoding

DESCRIÇÃO DO MODELO



MLP e CNN

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=10, stratify=y)
```

Divisão do conjunto de dados em 2 subconjuntos (train e test)

MLP

```
scaler = MinMaxScaler().fit(X_train)
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled  = scaler.transform(X_test)
```

Normalização dos dados de treino

```
from sklearn.neural_network import MLPClassifier
model = MLPClassifier(
    solver="lbfgs",
    activation = "relu",
    hidden_layer_sizes=(100,),
    max_iter=100,
    validation_fraction=0.1,
    random_state=10
)
print("Treinar o modelo...")
model.fit(X_train_scaled, y_train)
```

Criação e treino da rede MLP

CNN

```
model = Sequential([
    Conv2D(16, (3,3), activation='relu', input_shape=(32,32,3)),
    Flatten(),
    Dense(100, activation='relu'),
    Dense(5, activation='softmax')
])
```

```
modelo = model.fit(
    X_train, yTrain_binario,
    epochs=15,
    batch_size=16,
    validation_split=0.1,
    verbose=2)
```

Criação e treino da rede CNN

CNN

```
model.compile(optimizer='adam', loss='categorical_crossentropy')
```

Compilação e minimização do erro de classificação

DESCRIÇÃO DO MODELO

MLP

```
from sklearn.metrics import (
    accuracy_score, recall_score, f1_score,
    confusion_matrix, roc_auc_score
)

y_pred = model.predict(X_test_scaled)
y_prob = model.predict_proba(X_test_scaled)
acc = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred, average="macro")
f1 = f1_score(y_test, y_pred, average="macro")
auc = roc_auc_score(y_test, y_prob, multi_class="ovr")
```

Previsão e avaliação de desempenho

MLP e CNN

```
matriz = confusion_matrix(y_test, y_pred)
specificity = []
for i in range(len(matriz)):
    tn_i = matriz.sum() - (matriz[i, :].sum() + matriz[:, i].sum() - matriz[i, i])
    fp_i = matriz[:, i].sum() - matriz[i, i]
    specificity.append(tn_i / (tn_i + fp_i))
specificity_mean = np.mean(specificity)
```

Avaliação da sensibilidade

CNN

```
from sklearn.metrics import accuracy_score, recall_score, f1_score, confusion_matrix, roc_auc_score

y_prob = model.predict(X_test)
y_pred = np.argmax(y_prob, axis=1)
acc = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred, average="macro")
f1 = f1_score(y_test, y_pred, average="macro")
auc = roc_auc_score(yTest_binario, y_prob, multi_class="ovr")
```

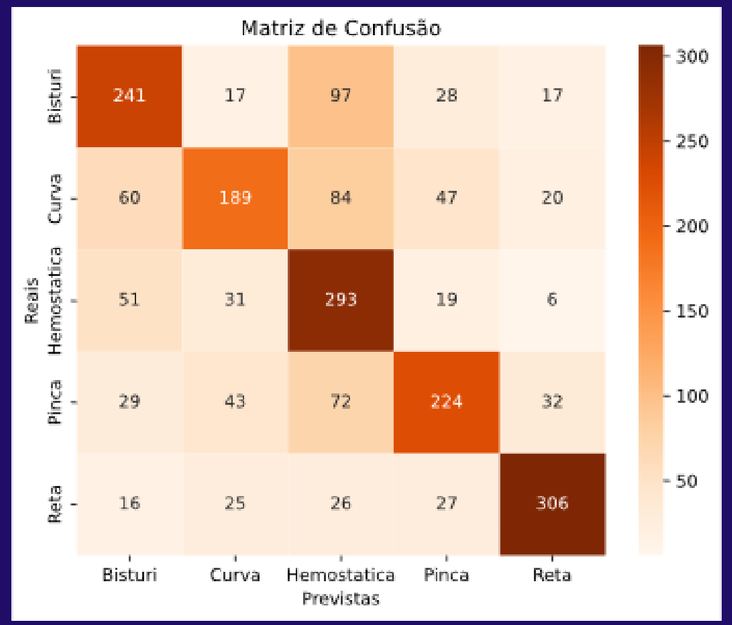
Previsão e avaliação de desempenho

ANÁLISE DE RESULTADOS

MLP

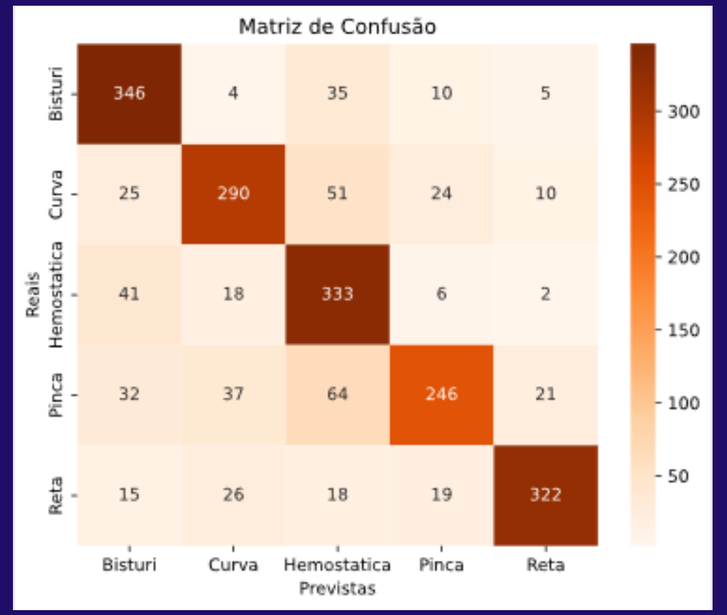
Resultados MLP com função lbfgs, camada de 100 neurónios e 100 iterações

```
Accuracy: 0.626
Recall: 0.627
Especificidade: 0.907
F1-score: 0.626
AUC: 0.866
```



Resultados MLP com função adam, camada de 100 neurónios e 100 iterações

```
Accuracy: 0.768
Recall: 0.768
Especificidade: 0.942
F1-score: 0.768
AUC: 0.948
```

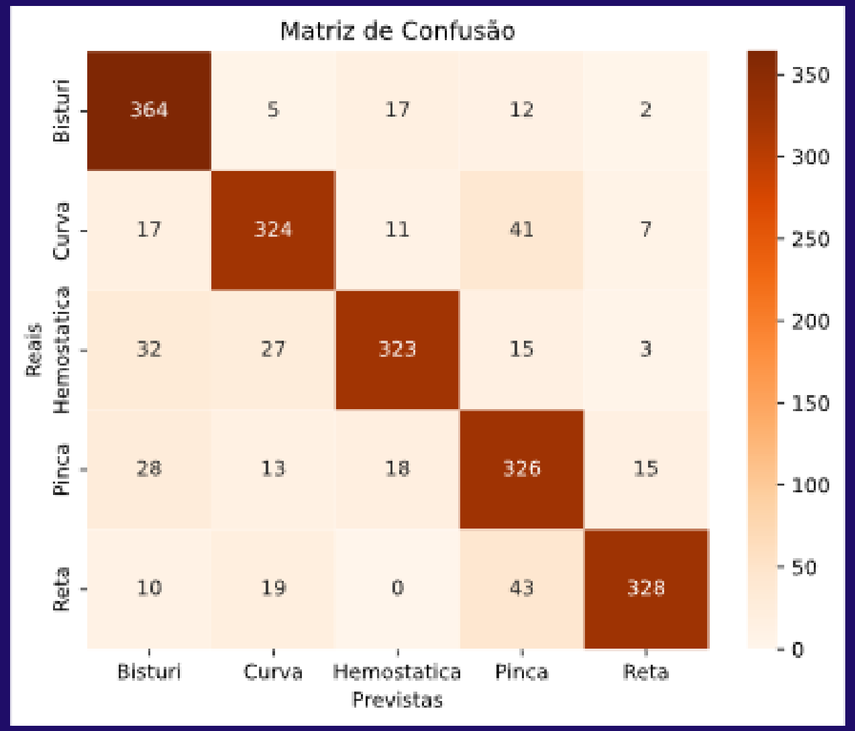


ANÁLISE DE RESULTADOS

MLP

Resultados MLP com função adam e com 2 camadas de 100,200 neurónios e 500 iterações

```
Accuracy: 0.833
Recall: 0.833
Especificidade: 0.958
F1-score: 0.833
AUC: 0.969
```



ANÁLISE DE RESULTADOS

CNN

Configurações CNN

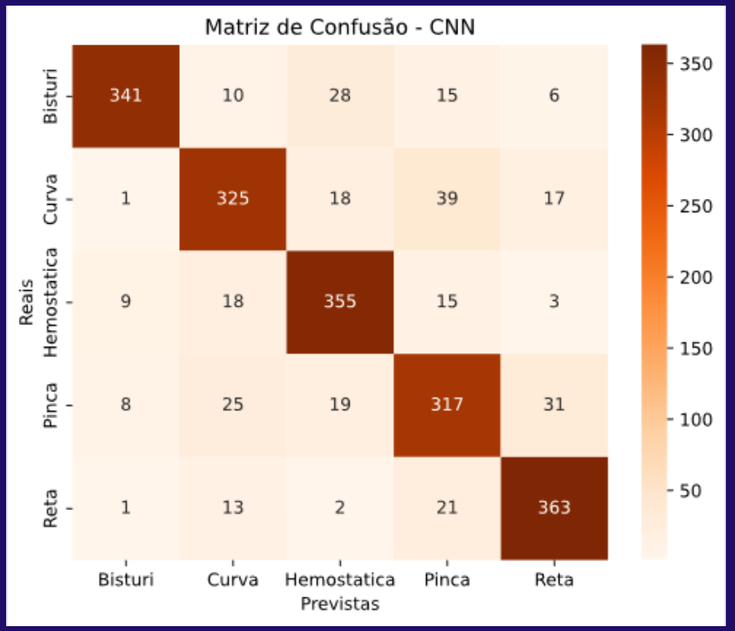
```
model = Sequential([
    Conv2D(16, (3,3),
    Flatten(),
    Dense(100, activation='relu'),
    Dense(5, activation='softmax')])
```

```
modelo_treino = model.fit(
    X_train, yTrain_binario,
    epochs=15,
    batch_size=16,
    validation_split=0.1,
    verbose=2)
```

Resultados CNN

```
Accuracy: 0.851
Recall: 0.851
F1-score: 0.851
Especificidade: 0.963
AUC: 0.974
```

```
**** Métricas por classe ****
Recall(Bisturi)=0.853
Recall(Curva)=0.812
Recall(Hemostatica)=0.887
Recall(Pinca)=0.792
Recall(Reta)=0.907
```



CONCLUSÃO

- A CNN apresentou um melhor desempenho, com 0.851 de accuracy e um AUC acima de 0.90 (o que deixa espaço para melhorias)

- Verificou-se alguma confusão entre classes semelhantes, como a Mayo Curva e a Pinça

- O sistema mostra potencial clínico, podendo tornar-se fiável com uma otimização.