Pipeline de Visão Computacional e Inteligência Artificial

Nos últimos anos, aplicações de Visão Computacional (VC) e Inteligência Artificial (IA) vêm crescendo rapidamente em diversos setores: medicina (diagnóstico por imagem), indústria (inspeção automatizada), segurança (reconhecimento facial), entre outros.

Este trabalho visa desenvolver e documentar um sistema prático que atende: processamento de imagens, detecção e segmentação, extração de características, transformações geométricas e aplicação de IA/ML com uma rede neural convolucional (CNN) em MNIST.

por Bipe Pinheiro e Yann Lucas

OpenCV

TensorFlow

Principais Bibliotecas Utilizadas

OpenCV (cv2)

Plataforma de visão computacional. Oferece funções para leitura/escrita de imagens, filtros, conversão de cores, equalização de histograma, morfologia, detecção de bordas e extração de características.

NumPy

Usado para manipulação de arrays e matrizes. Essencial para criação de kernels de filtro, matrizes de transformação e conversão de gradientes.

TensorFlow/Keras

Framework de aprendizado profundo para definir, compilar, treinar e avaliar a rede neural convolucional (CNN). Inclui submódulos para montagem da arquitetura, data augmentation e callbacks.

Pré-processamento de Imagens

O pré-processamento de imagens é uma etapa crucial em qualquer sistema de visão computacional. Antes de extrair features ou segmentar objetos, é fundamental reduzir ruído, equalizar contrastes e padronizar o tamanho espacial das imagens.

Leitura e Redimensionamento

Carregamento das imagens e redimensionamento para um tamanho padrão (ex: 256x256 pixels).

Filtro Gaussian Blur

Redução de ruídos de alta frequência, suavizando a imagem.

Filtro de Sharpening

Realce de arestas e detalhes, tornando a imagem mais nítida.

Equalização de Histograma

Redistribuição dos níveis de intensidade para melhorar o contraste da imagem.

```python
# ==========================
# 1. Carregar e redimensionar todas as imagens para tamanho comum
# ==========================
nomes_arquivos = [f'Teste/img{i}.jpg' for i in range(1, 11)]
imagens_originais = []
for caminho in nomes_arquivos:
    img = cv2.imread(caminho)
    if img is not None:
        # Redimensionar para 256x256 já ao carregar
        img_red = cv2.resize(img, (256, 256))
        imagens_originais.append(img_red)
    else:
        print(f"Não foi possível carregar: {caminho}")

if not imagens_originais:
    raise SystemExit("Nenhuma imagem foi carregada. Verifique os nomes e caminhos.")

# Exibir todas as imagens originais em grade
show_grid(imagens_originais, 'Imagens Originais (256x256)')

# ==========================
# 2. Pré-processamento: Blur e Sharpen
# ==========================
# Gaussian Blur
imagens_blur = [cv2.GaussianBlur(img, (5, 5), 0) for img in imagens_originais]
show_grid(imagens_blur, 'Blur (Gaussian)')

# Sharpening
kernel_sharp = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
imagens_sharp = [cv2.filter2D(img, -1, kernel_sharp) for img in imagens_originais]
show_grid(imagens_sharp, 'Sharpen')

# ==========================
# 3. Conversão para Cinza e Equalização de Histograma
# ==========================
# Conversão para escala de cinza
imagens_cinza = [cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) for img in imagens_originais]
show_grid(imagens_cinza, 'Imagens em Tons de Cinza')

# Equalização de Histograma
imagens_eq = [cv2.equalizHist(img) for img in imagens_cinza]
show_grid(imagens_eq, 'Equalização de Histograma')
```

Detecção e Segmentação de Imagens

Após o pré-processamento, a próxima etapa é segmentar regiões de interesse e detectar bordas, preparando a imagem para extração de características mais avançadas ou para delimitar contornos de objetos.

Thresholding Fixo

Converte uma imagem em tons de cinza em uma imagem binária (0 ou 255), de acordo com um limiar fixo. Se o pixel é maior que o limiar, assume valor 255 (branco); caso contrário, assume 0 (preto).

Em imagens com distribuição de pixel aproximada uniforme, o valor mediano (127) pode funcionar razoavelmente para segmentar regiões médias.

Detecção de Bordas com Canny

O detector de bordas Canny é um algoritmo clássico em visão computacional, considerado robusto e de baixo custo computacional. Ele consiste em suavização, cálculo de gradiente, aplicação de não-máxima supressão e histerese com dois thresholds.

Canny produz contornos nítidos e costuma ser usado em pipelines que necessitam de segmentação precisa.

Detecção de Bordas com Sobel

O filtro de Sobel calcula as derivadas parciais da imagem nas direções X e Y, gerando dois mapas de gradiente (sobelX e sobelY). A combinação destes (magnitude) mostra as regiões de maior variação de intensidade, correspondendo a bordas.

Computar derivada na direção X

Utilizando cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3) para calcular a derivada horizontal

Computar derivada na direção Y

Utilizando cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3) para calcular a derivada vertical

Calcular magnitude do gradiente

Usando cv2.magnitude(sobelx, sobely) para obter a intensidade total da variação

Converter para formato adequado

Transformando para uint8 com np.uint8(np.clip(sobel_mag, 0, 255))

```python
# ==========================
# 4. Segmentação (Thresholding)
# ==========================
segmentadas = [cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)[1] for img in imagens_cinza]
show_grid(segmentadas, 'Segmentação (Thresholding)')


# ==========================
# 5. Detecção de Bordas: Canny e Sobel
# ==========================
# 5.1 Canny
bordas_canny = [cv2.Canny(img, 100, 200) for img in imagens_cinza]
show_grid(bordas_canny, 'Bordas Canny')

# 5.2 Sobel
bordas_sobel = []
for img in imagens_cinza:
    sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
    sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)
    sobel_mag = cv2.magnitude(sobelx, sobely)
    sobel_uint8 = np.uint8(np.clip(sobel_mag, 0, 255))
    bordas_sobel.append(obel_uint8)
show_grid(bordas_sobel, 'Bordas Sobel')
```

Extração de Características com ORB

A extração de características consiste em identificar pontos-chave ("keypoints") e calcular descritores associados que descrevem a vizinhança desses keypoints. Esses descritores podem ser comparados entre duas imagens para encontrar correspondências.

FAST Detector

Identifica candidatos a keypoints baseados em comparações de pixel em círculos de raio 3, detectando pontos de interesse na imagem.

BRIEF Descriptor

Gera vetores de bits (0/1) com testes de comparações diretas de intensidades de pares de pixels em uma janela local ao redor do keypoint.

Invariância a Rotação

ORB estima a orientação dominante de cada keypoint e rotaciona a janela local antes de extrair o descritor BRIEF, garantindo invariância a rotação.

```python
# ==========================
# 6. Extração de Características (ORB) e exibição dos keypoints
# ==========================
orb = cv2.ORB_create()
imagens_orb = []  # imagens com keypoints desenhados
descritores_lista = []
keypoints_lista = []
for img_color, img_gray in zip(imagens_originais, imagens_cinza):
    kp, des = orb.detectAndCompute(img_gray, None)
    descritores_lista.append(des)
    keypoints_lista.append(kp)
    img_kp = cv2.drawKeypoints(img_color, kp, None, color=(0, 255, 0))
    imagens_orb.append(img_kp)
show_grid(imagens_orb, 'Keypoints ORB')
```

Comparação de Descritores com BFMatcher

Para demonstrar como comparar descritores de duas imagens e buscar correspondências, utilizamos o BFMatcher (Brute-Force Matcher) do OpenCV, que compara cada descritor da primeira imagem com cada descritor da segunda.

Criar o matcher

cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

Encontrar correspondências

matches = bf.match(des1, des2)

Ordenar por similaridade

sorted(matches, key=lambda x: x.distance)

Visualizar matches

cv2.drawMatches(...)

Transformações Geométricas

Em muitas aplicações, objetos ou cenas podem aparecer em diferentes orientações, escalas ou posições. Para testar a robustez de algoritmos, realizamos transformações geométricas controladas.

Rotação

Utiliza cv2.getRotationMatrix2D e cv2.warpAffine
para girar a imagem em torno de um ponto central.
Útil para testar invariância a orientação.

Escala

Aplica cv2.resize com fatores fx e fy
para aumentar ou diminuir o tamanho da imagem.
Testa a invariância a mudanças de tamanho.

Translação

Cria uma matriz de translação e aplica cv2.warpAffine
para mover a imagem horizontal e verticalmente.
Simulando deslocamentos.

```python
# ========================
# 7. Transformações Geométricas: Rotação, Escala e Translação
# ========================
# Rotação (45 graus)
imagens_rot = []
h, w = imagens_originais[0].shape[:2]
centro = (w // 2, h // 2)
for img in imagens_originais:
    mat_rot = cv2.getRotationMatrix2D(centro, 45, 1.0)
    img_rot = cv2.warpAffine(img, mat_rot, (w, h))
    imagens_rot.append(img_rot)
show_grid(imagens_rot, 'Rotação 45°')

# Escala (1.2x)
# Mostrar dimensão antes, depois da escala e após ajuste para 256x256
imagens_esc = []
imagens_escaj = []
for idx, img in enumerate(imagens_originais):
    print(f"Imagem {idx+1} - Forma original: {img.shape}")
    img_scaled = cv2.resize(img, None, fx=1.2, fy=1.2)
    print(f"Imagem {idx+1} - Forma após escala 1.2x: {img_scaled.shape}")
    img_resized = cv2.resize(img_scaled, (256, 256))
    print(f"Imagem {idx+1} - Forma após ajuste para 256x256: {img_resized.shape}\n")
    imagens_esc.append(img_scaled)
    imagens_escaj.append(img_resized)
show_grid(imagens_escaj, 'Escala 1.2x (Ajustada para 256x256)')

# Translação (+50 px em x, +30 px em y)
imagens_trans = []
mat_trans = np.float32([[1, 0, 50], [0, 1, 30]])
for img in imagens_originais:
    img_tr = cv2.warpAffine(img, mat_trans, (w, h))
    imagens_trans.append(img_tr)
show_grid(imagens_trans, 'Translação (+50, +30)')

print("Processamento de Imagens concluído.")
```

Aplicação de IA/ML - CNN em MNIST

Para a fase de IA/ML, após o pré-processamento de imagens reais, escolhemos o dataset MNIST (Dígitos Manuscritos) para a classificação supervisionada. A simplicidade e padronização do MNIST são vantajosas para a didática de redes neurais, e essa escolha permite focar a demonstração de aprendizado profundo separadamente do pré-processamento de imagens coloridas.

Motivação e Escolha do Dataset

Simplicidade e Padronização

Possuindo apenas 10 classes (0 a 9) e imagens em tons de cinza 28×28, permitindo foco na arquitetura e métrica.

Integração com Keras

O dataset já vem integrado ao Keras, facilitando o carregamento imediato para o desenvolvimento do modelo.

Características do Dataset

MNIST possui 60.000 imagens de treino e 10.000 de teste (28×28, grayscale, valores 0-255), com rótulos correspondentes aos dígitos manuscritos (0-9).

Pré-processamento do MNIST

Carregamento dos Dados

Utilizamos a função tf.keras.datasets.mnist.load_data() que baixa e carrega MNIST, retornando tuplas.

Reshape para Compatibilidade

Aplicamos reshape (-1, 28, 28, 1) para inserir um canal extra, tornando os dados compatíveis com camadas Conv2D do Keras.

Normalização dos Valores

Convertemos os valores de inteiro 0-255 para ponto flutuante 0.0-1.0 com astype('float32') / 255.0, acelerando e estabilizando o treinamento.

```python
# ------------------------------------------------
# Carrega o conjunto de dados MNIST diretamente do Keras
# ------------------------------------------------
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()


# ------------------------------------------------
# Precisamos:
#   1) Adicionar a dimensão de canal (canal único, shape → 28×28×1).
#   2) Normalizar os pixels para a faixa [0.0, 1.0] dividindo por 255.
# ------------------------------------------------
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
x_test  = x_test.reshape( -1, 28, 28, 1).astype('float32') / 255.0

# Verificação rápida dos shapes
print("x_train.shape:", x_train.shape)
print("x_test.shape: ", x_test.shape)

# Labels já vêm como inteiros de 0 a 9
print("y_train.shape:", y_train.shape)
print("y_test.shape: ", y_test.shape)
```

Arquitetura da Rede Neural Convolucional

```python
model = models.Sequential([
    # Primeiro bloco conv + pool
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),

    # Segundo bloco conv + pool
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),

    # Flatten + Fully Connected
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')  # 10 classes (dígitos 0–9)
])
```

Bloco Convolucional 1

- **Conv2D(32, (3,3), activation='relu'):** Encontra padrões na imagem usando 32 "filtros" de 3×3 pixels
- **BatchNormalization():** Estabiliza o aprendizado, ajustando a ativação dos neurônios.
- **MaxPooling2D((2, 2)):** Reduz dimensões de 28×28 para 14×14.
- **Dropout(0.25):** Evita "memorize" nos dados de treino, desativando aleatoriamente 25% dos neurônios durante o treinamento.

Bloco Convolucional 2

Similar ao Bloco 1, mas com 64 filtros de 3×3. Reduz dimensões de 14×14 para 7×7. Aumenta a capacidade de extração de características mais complexas.

Flatten

Achata os mapas de características 7×7×64 em um vetor de tamanho 7×7×64=3136, preparando para as camadas densas.

Camadas Densas

Dense(128, activation='relu') seguida de Dropout(0.5) e Dense(10, activation='softmax') para classificação final dos 10 dígitos.

Configuração de Callbacks

```python
# EarlyStopping

early_stop = EarlyStopping(
    monitor='val_loss',          # Monitora a perda de validação
    patience=5,                  # Se não melhorar em 3 épocas, para
    restore_best_weights=True,   # Restaura pesos da melhor época
    verbose=1
)

# Lista de callbacks que serão passados ao fit
callbacks = [early_stop]
```

EarlyStopping

Monitora a métrica de perda de validação (val_loss) e interrompe o treino caso não haja melhora após um número definido de épocas ("patience" = 5). Restaura os melhores pesos encontrados durante o treinamento.

Motivação

Evitar overfitting: quando o modelo não melhora mais em validação, continuar treinando tende a "decorar" o conjunto de treino sem ganhar generalização. Garantir que os pesos que apresentaram melhor desempenho em validação sejam mantidos.

Benefícios

Economiza tempo de processamento ao interromper o treinamento quando não há mais ganhos significativos.
Simplifica o processo, já que a alteração manual de learning rate não será necessária.

Optimizador

Optimizer = “adam”

Otimizador adaptativo que combina RMSProp e momentum, geralmente funciona bem sem ajuste fino de learning rate.

Loss= 'sparse_categorical_crossentropy'

Adequado para rótulos em forma de inteiros (0–9), internamente converte para codificação one-hot.

metrics= ['accuracy']

monitora acurácia de treino/validação durante fit.

model.summary()

Exibe no console a arquitetura completa, mostrando a forma de saída de cada camada e número de parâmetros treináveis.

```
# ------------------------------------------------
# Compila o modelo definindo:
#   - Otimizador: 'adam' (com taxa de aprendizado adaptativa)
#   - Função de perda: sparse_categorical_crossentropy (rótulos são inteiros)
#   - Métrica: 'accuracy' para acompanhar acurácia de treino e validação
# ------------------------------------------------
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# Mostra resumo do modelo (número de parâmetros, saída de cada layer, etc.)
model.summary()
```

Treinamento da CNN

Configuração de Batch

Definimos batch_size=128, equilibrando uso de memória e eficiência de treino. Este valor intermediário é adequado para GPUs, sendo uma potência de 2.

Histórico

O objeto history retornado por model.fit contém dicionários com métricas por época, permitindo análise posterior do treinamento.

Validação

Utilizamos validation_data=(x_test, y_test) para computar métricas de validação (loss e accuracy) no conjunto de teste a cada época.

Épocas

Definimos epochs=40 como número máximo, embora o EarlyStopping provavelmente interrompa antes se val_loss deixar de cair.

Avaliação Final e Métricas

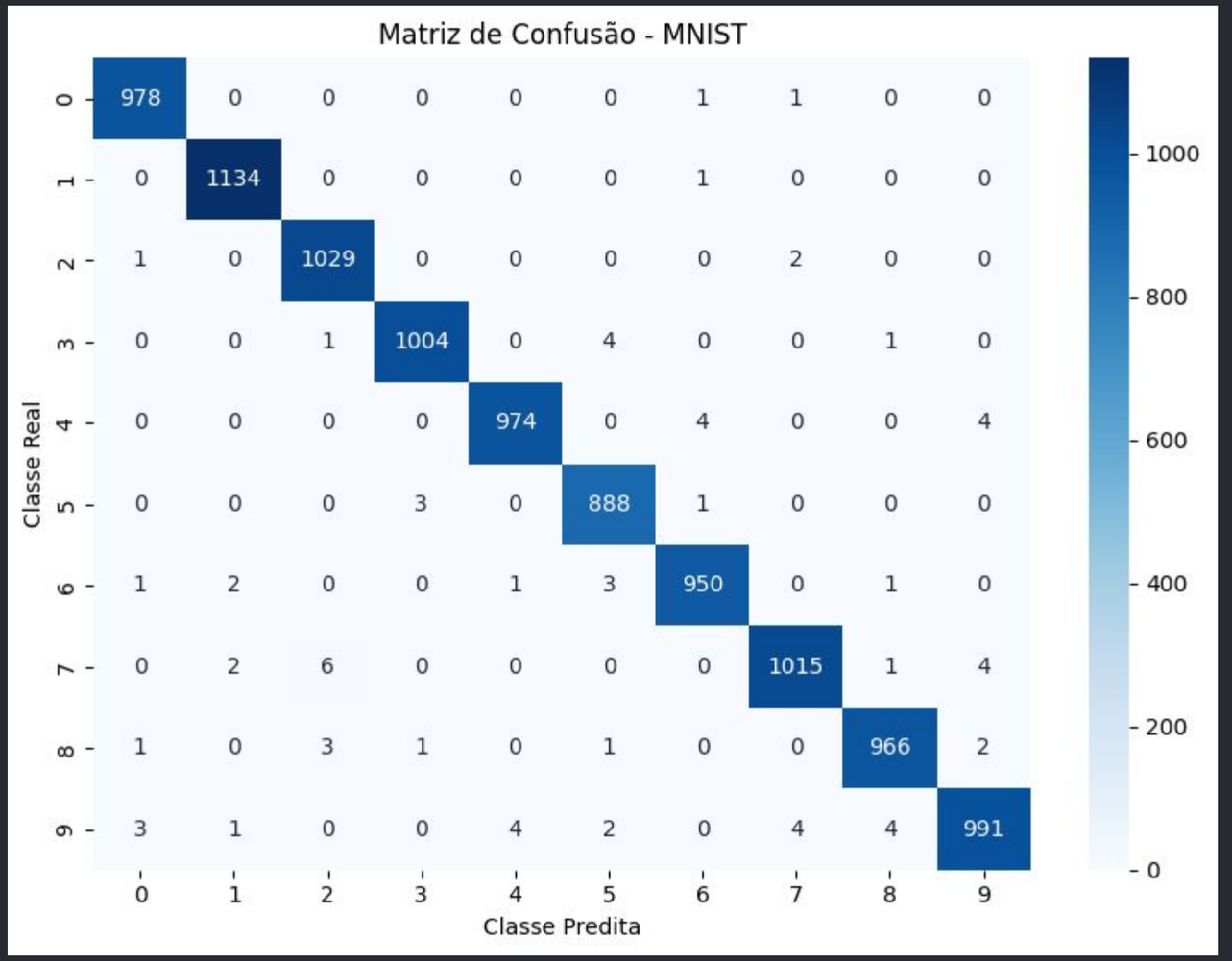
Avaliação no Teste

Utilizamos model.evaluate(x_test, y_test) para calcular as métricas definidas em compile() (loss e accuracy) no conjunto de teste, obtendo test_loss (float) e test_acc (float).
A predição é feita com model.predict(x_test), gerando um array de forma (10000, 10), onde cada linha é uma distribuição de probabilidade.

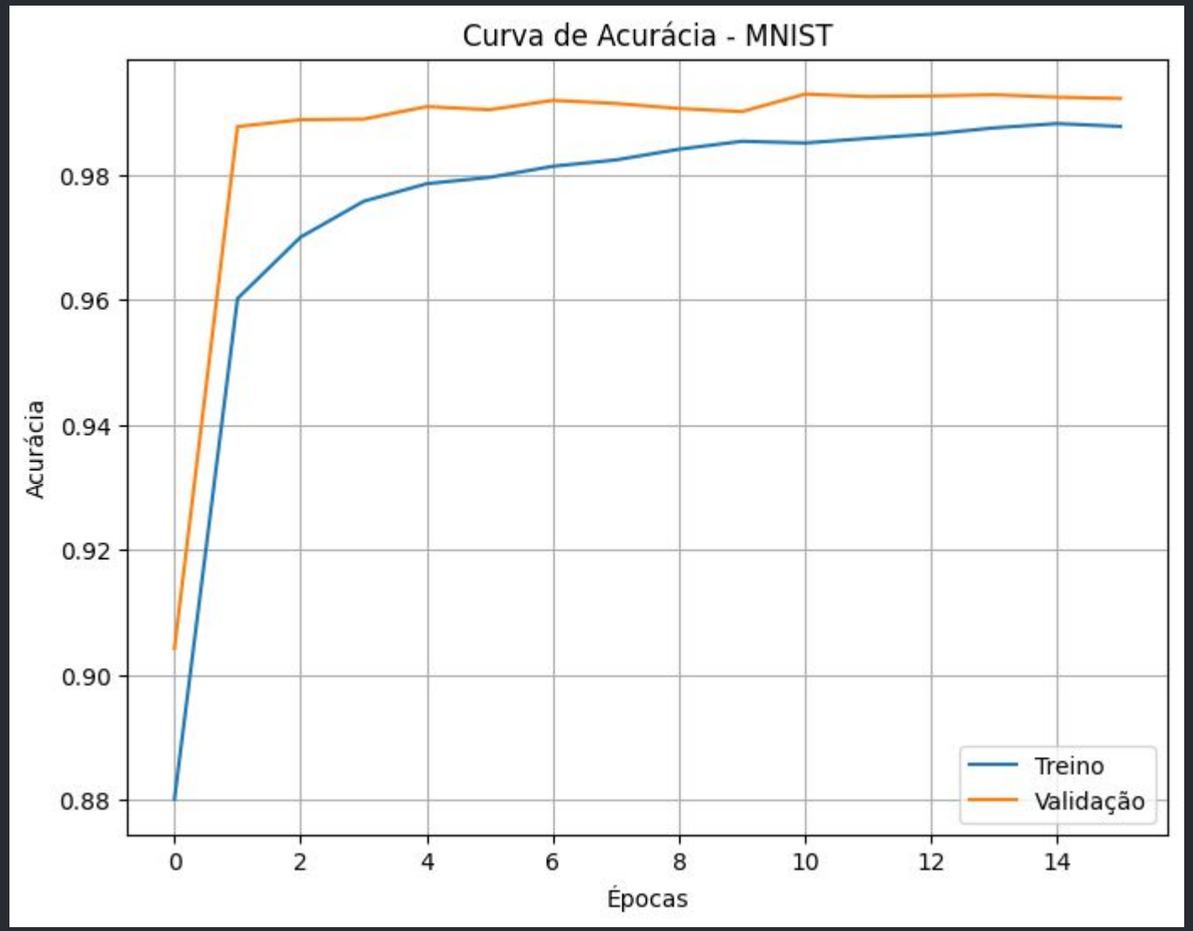
Matriz de Confusão

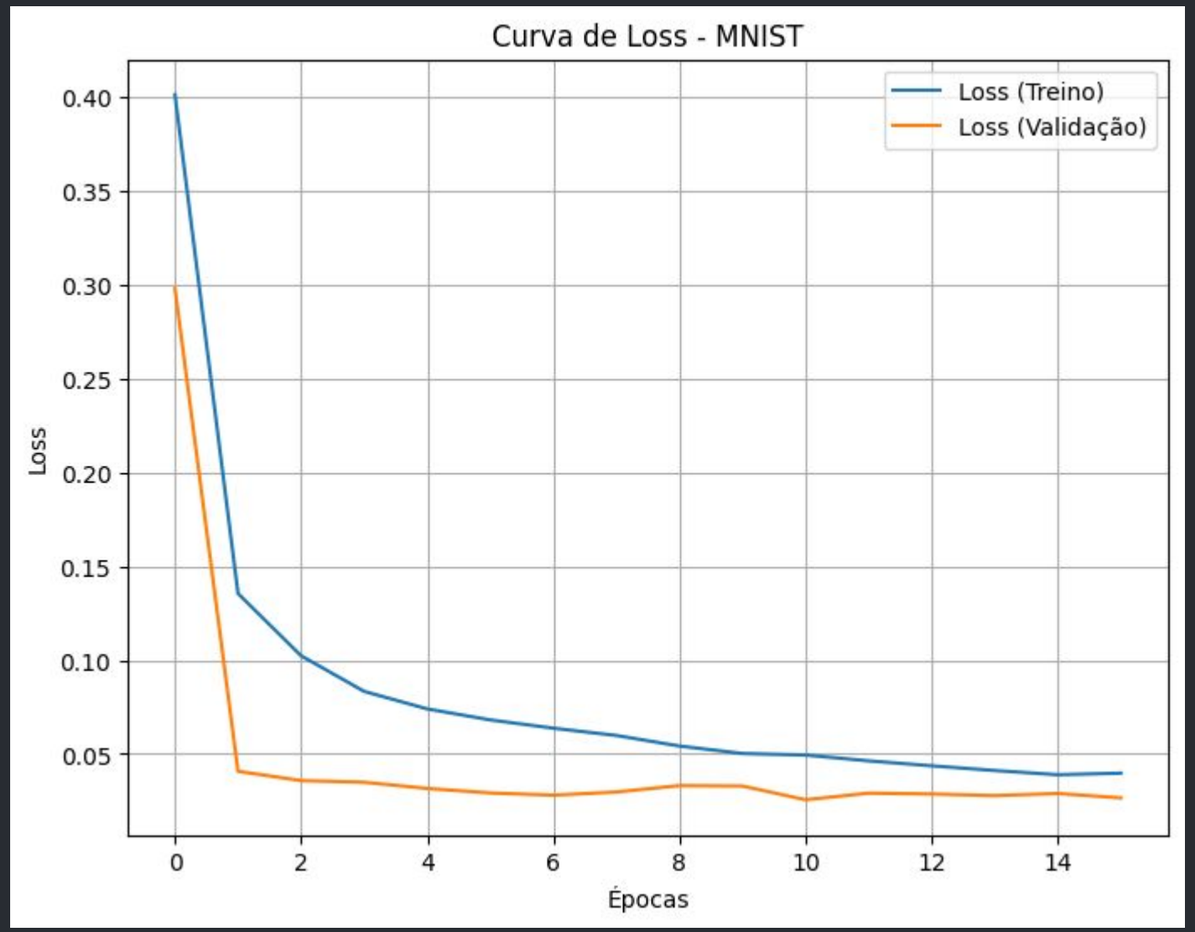
Aplicamos confusion_matrix(y_test, y_pred) para gerar uma matriz 10×10, onde a linha i representa as ocorrências reais da classe i, e a coluna j representa quantas dessas foram preditas como classe j.
Elementos da diagonal principal correspondem a acertos; elementos fora da diagonal, a erros.

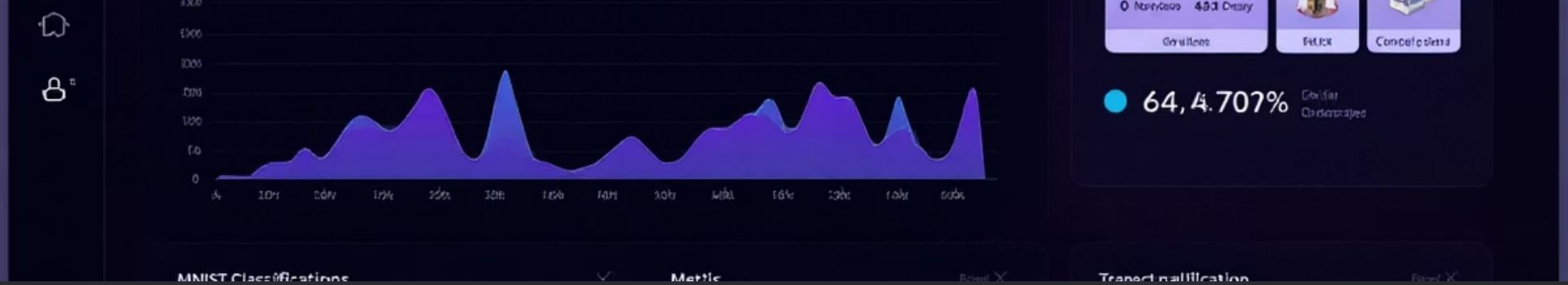
```
Matriz de confusão:
 [[ 978    0    0    0    0    0    1    1    0    0]
 [   0 1134    0    0    0    0    1    0    0    0]
 [   1    0 1029    0    0    0    0    2    0    0]
 [   0    0    1 1004    0    4    0    0    1    0]
 [   0    0    0    0  974    0    4    0    0    4]
 [   0    0    0    3    0  888    1    0    0    0]
 [   1    2    0    0    1    3  950    0    1    0]
 [   0    2    6    0    0    0    0 1015    1    4]
 [   1    0    3    1    0    1    0    0  966    2]
 [   3    1    0    0    4    2    0    4    4  991]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       980
           1       1.00      1.00      1.00      1135
           2       0.99      1.00      0.99      1032
           3       1.00      0.99      1.00      1010
           4       0.99      0.99      0.99       982
           5       0.99      1.00      0.99       892
           6       0.99      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       0.99      0.99      0.99       974
           9       0.99      0.98      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```



Visualização de Desempenho







Resultados Obtidos

99.15%

Acurácia Final

Desempenho excepcional no conjunto de teste

0.0307

Loss Final

Valor baixo indicando alta confiança nas predições

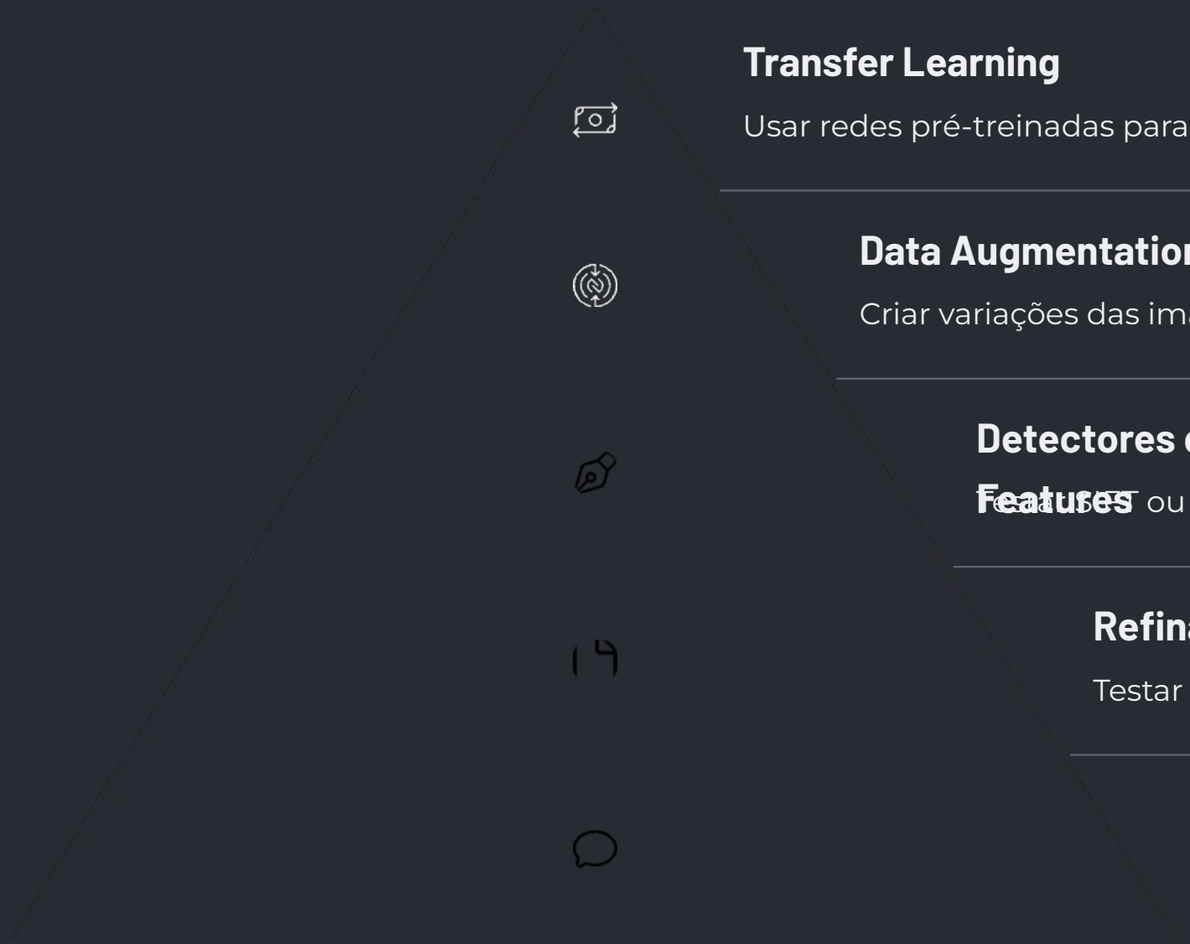
0.99

F1-Score Médio

Equilíbrio entre precisão e recall

A CNN alcançou excelentes resultados, com precisão (precision) e recall acima de 0.98 para todas as classes. A classe 7 teve F1-score um pouco menor (0.98-0.99), indicando alguns erros residuais (possivelmente 7 confundido com 9 ou 1). No geral, o desempenho foi homogêneo entre todas as classes.

Limitações e Melhorias Futuras



Transfer Learning

Usar redes pré-treinadas para imagens reais

Data Augmentation

Criar variações das imagens, para aumentar o dataset e melhorar a generalização.

Detectores de Features

Testar SIFT ou SURF para maior robustez

Refinamento de Hiperparâmetros

Testar arquiteturas mais profundas e otimizadores

Learning Rate Schedule

Ajustar a taxa de aprendizado durante o treino para otimizar a performance.

Embora cada módulo tenha entregado resultados satisfatórios, existem limitações inerentes como sensibilidade de thresholds e generalização da CNN para imagens coloridas. As melhorias sugeridas permitiriam estender o projeto para cenários mais complexos e aplicações práticas.