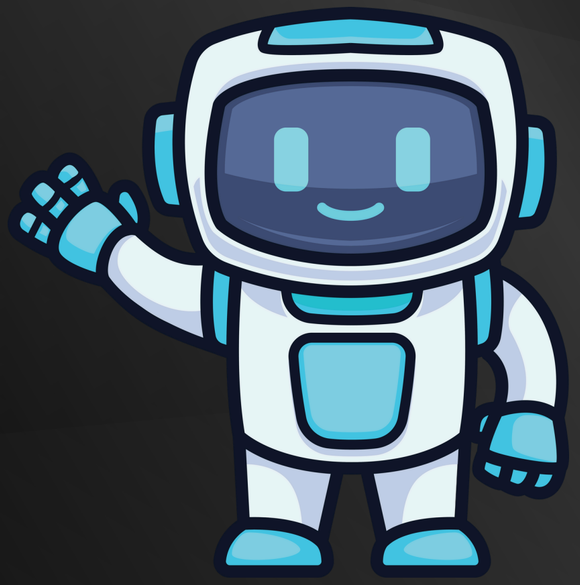
REDE CNN COM DATASET MNIST



Abel Junior, Eri Tiecher, Karine Haubert, Patrick Dutra, Mateus Miri,
Matheus Maschio, Benhur Machado e Eduardo Rossatto

OQUE É AS CNNS

Redes Neurais Convolucionais (CNN):

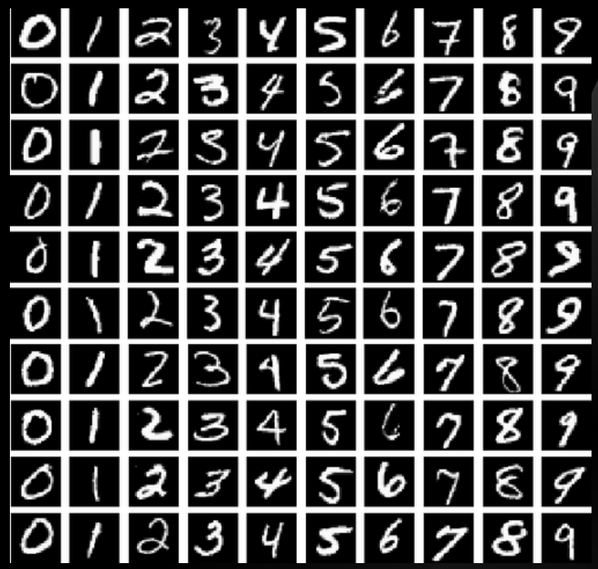
As redes neurais convolucionais (CNNs) são uma classe especializada de redes neurais profundas projetadas para processar dados com estrutura de grade, como imagens. Elas são amplamente usadas em tarefas de visão computacional, como classificação de imagens, detecção de objetos e segmentação semântica.

PONTOS-CHAVE SOBRE CNNS:

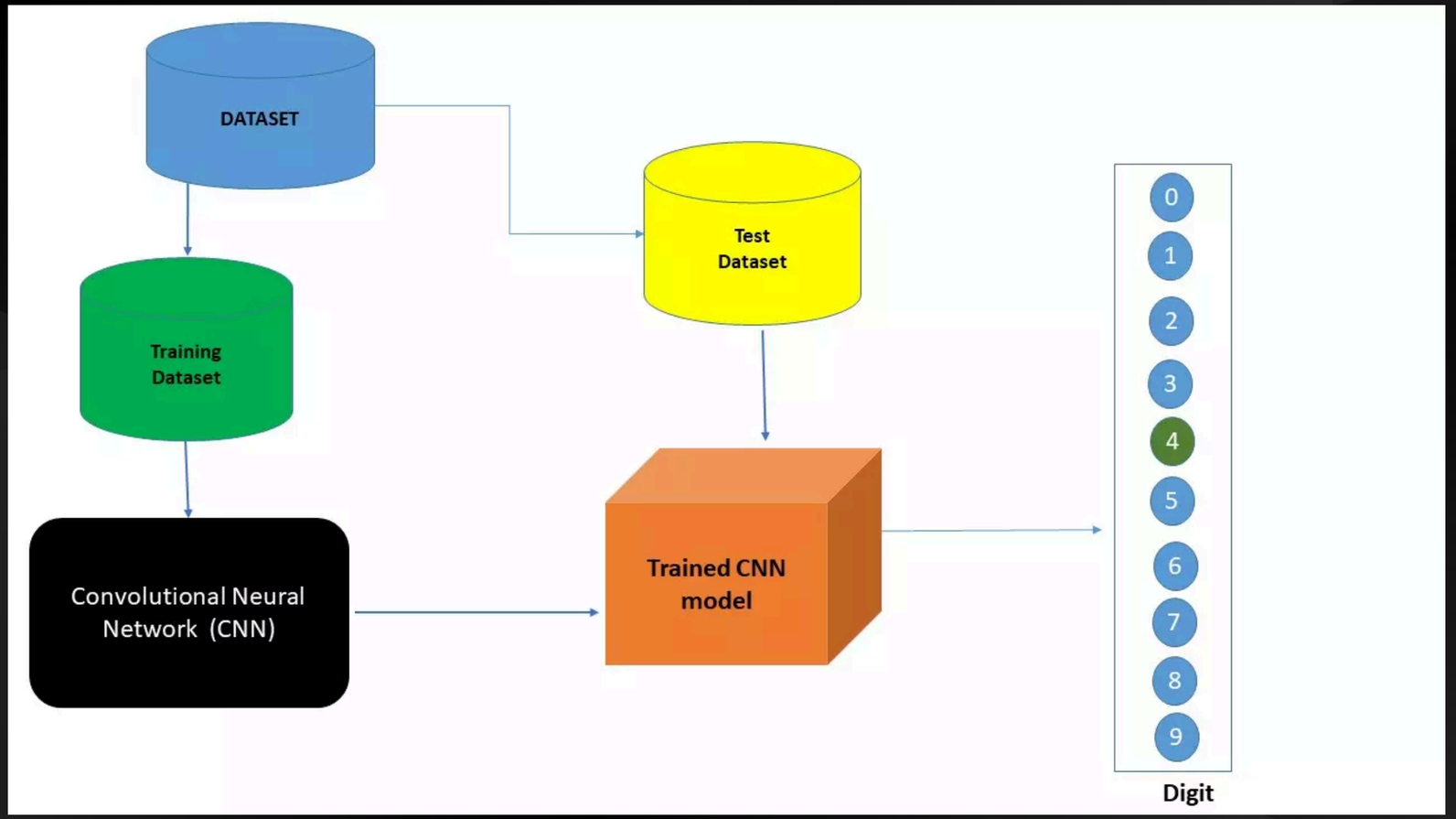
- Camadas convolucionais
- Pooling layer
- Camadas totalmente conectadas

DATASET MNIST

O conjunto de dados MNIST é um dos mais populares na área de aprendizado de máquina. Ele consiste em 70.000 imagens em escala de cinza de dígitos manuscritos (0 a 9), cada uma com tamanho de 28x28 pixels. O objetivo é classificar corretamente os dígitos com base nas imagens

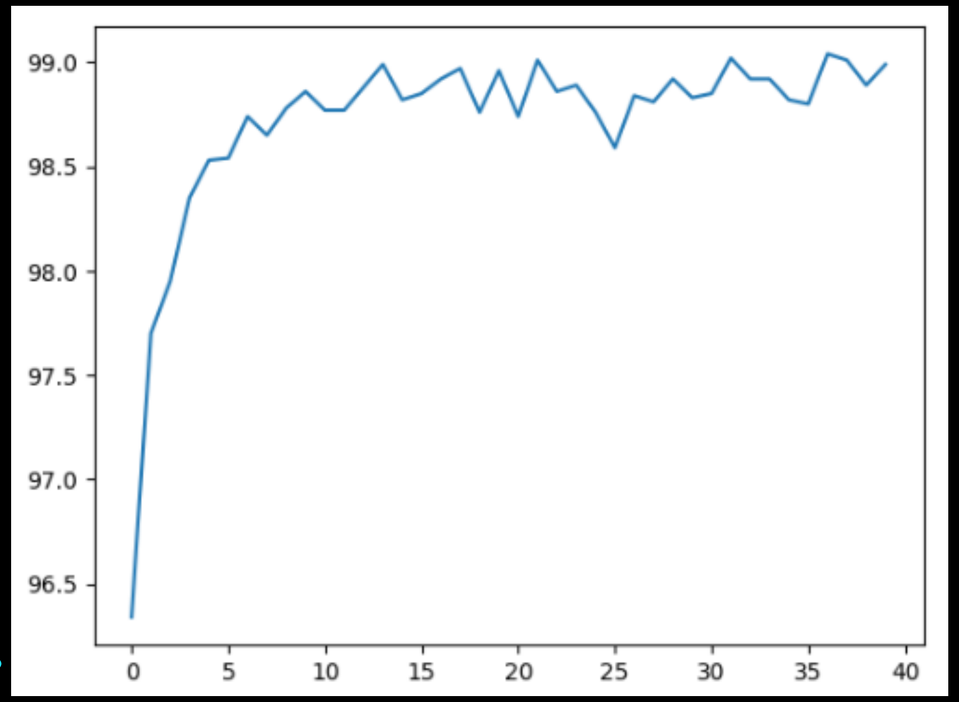


MODELO


DATASET
Test Dataset
Training Dataset
Convolutional Neural Network (CNN)
Trained CNN model
0
1
2
3
4
5
6
7
8
9
Digit

TREINAMENTO

- 40 Epoch´s
- Cada um com sua determinada Precisão
- Salva o modelo de melhor precisão baseado na maior precisão




```
Saving Best Model with Accuracy:  96.33999633789062
Epoch: 1 Accuracy : 96.33999633789062 %
Saving Best Model with Accuracy:  97.69999694824219
Epoch: 2 Accuracy : 97.69999694824219 %
Saving Best Model with Accuracy:  97.94999694824219
Epoch: 3 Accuracy : 97.94999694824219 %
Saving Best Model with Accuracy:  98.3499984741211
Epoch: 4 Accuracy : 98.3499984741211 %
Saving Best Model with Accuracy:  98.52999877929688
Epoch: 5 Accuracy : 98.52999877929688 %
Saving Best Model with Accuracy:  98.54000091552734
Epoch: 6 Accuracy : 98.54000091552734 %
Saving Best Model with Accuracy:  98.73999786376953
Epoch: 7 Accuracy : 98.73999786376953 %
Epoch: 8 Accuracy : 98.6500015258789 %
Saving Best Model with Accuracy:  98.77999877929688
Epoch: 9 Accuracy : 98.77999877929688 %
Saving Best Model with Accuracy:  98.86000061035156
Epoch: 10 Accuracy : 98.86000061035156 %
Epoch: 11 Accuracy : 98.7699966430664 %
Epoch: 12 Accuracy : 98.7699966430664 %
Saving Best Model with Accuracy:  98.87999725341797
Epoch: 13 Accuracy : 98.87999725341797 %
Saving Best Model with Accuracy:  98.98999786376953
Epoch: 14 Accuracy : 98.98999786376953 %
Epoch: 15 Accuracy : 98.81999969482422 %
Epoch: 16 Accuracy : 98.8499984741211 %
Epoch: 17 Accuracy : 98.91999816894531 %
Epoch: 18 Accuracy : 98.97000122070312 %
Epoch: 19 Accuracy : 98.76000213623047 %
Epoch: 20 Accuracy : 98.95999908447266 %
Epoch: 21 Accuracy : 98.73999786376953 %
Saving Best Model with Accuracy:  99.01000213623047
Epoch: 22 Accuracy : 99.01000213623047 %
Epoch: 23 Accuracy : 98.86000061035156 %
Epoch: 24 Accuracy : 98.88999938964844 %
Epoch: 25 Accuracy : 98.76000213623047 %
Epoch: 26 Accuracy : 98.58999633789062 %
Epoch: 27 Accuracy : 98.83999633789062 %
Epoch: 28 Accuracy : 98.80999755859375 %
Epoch: 29 Accuracy : 98.91999816894531 %
Epoch: 30 Accuracy : 98.83000183105469 %
Epoch: 31 Accuracy : 98.8499984741211 %
Saving Best Model with Accuracy:  99.0199966430664
Epoch: 32 Accuracy : 99.0199966430664 %
Epoch: 33 Accuracy : 98.91999816894531 %
Epoch: 34 Accuracy : 98.91999816894531 %
Epoch: 35 Accuracy : 98.81999969482422 %
Epoch: 36 Accuracy : 98.80000305175781 %
Saving Best Model with Accuracy:  99.04000091552734
Epoch: 37 Accuracy : 99.04000091552734 %
Epoch: 38 Accuracy : 99.01000213623047 %
Epoch: 39 Accuracy : 98.88999938964844 %
Epoch: 40 Accuracy : 98.98999786376953 %
```

VALIDAÇÃO

- Faz a predição com os dados de validação
- E monta uma matriz com as predições e valores corretos

```
*[104]: # Validando o Modelo
y_pred, y_true = predict_dl(lenet, val_dl)
pd.DataFrame(confusion_matrix(y_true, y_pred, labels=np.arange(0,10)))
```

[104]:

	0	1	2	3	4	5	6	7	8	9
0	974	1	2	0	0	0	0	1	2	0
1	0	1134	0	0	0	0	0	0	1	0
2	0	1	1028	0	1	0	0	1	1	0
3	0	1	4	999	0	2	0	1	3	0
4	0	0	1	0	973	0	3	0	0	5
5	2	1	0	6	0	878	1	0	3	1
6	1	3	0	0	3	1	950	0	0	0
7	1	2	5	0	1	0	0	1016	1	2
8	2	0	1	1	0	0	0	0	967	3
9	1	1	0	1	8	5	0	5	3	985

RESUMO DO CÓDIGO

```
import torch, torchvision
from torch import nn
from torch import optim
from torchvision.transforms import ToTensor
import torch.nn.functional as F
import matplotlib.pyplot as plt
import requests
from PIL import Image
from io import BytesIO
import copy
from sklearn.metrics import confusion_matrix
import pandas as pd
import numpy as np

numb_batch = 64
test_folder="./Train"
```

RESUMO DO CÓDIGO

```
# Instancia um Compose com diversos transforms, dentre eles:
    # Transforma a imagem para Tensor
    # Transforma a imagem para 28x28
    # Transforma a imagem para Grayscale (tons de cinza)
T = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Resize((28,28), antialias=True),
    torchvision.transforms.Grayscale(num_output_channels=1)
])

# Instancia Datasets Mnist para treinamento e validação
# Para isso, realiza o download do dataset
train_data = torchvision.datasets.MNIST('mnist_data', train=True, download=True, transform=T)
val_data = torchvision.datasets.MNIST('mnist_data', train=False, download=True, transform=T)

# Busca imagens do parâmetro root e adiciona no dataset de treinamento
# local_train_data = torchvision.datasets.ImageFolder(root=test_folder, transform=T)
# train_data= torch.utils.data.ConcatDataset([train_data, local_train_data])

# Instancia Dataloaders de Treinamento e Validação
train_dl = torch.utils.data.DataLoader(train_data, batch_size = numb_batch)
val_dl = torch.utils.data.DataLoader(val_data, batch_size = numb_batch)
```

RESUMO DO CÓDIGO

```python
# Método que define o Modelo
def create_lenet():
    model = nn.Sequential(
        nn.Conv2d(1, 6, 5, padding=2),
        nn.ReLU(),
        nn.AvgPool2d(2, stride=2),
        nn.Conv2d(6, 16, 5, padding=0),
        nn.ReLU(),
        nn.AvgPool2d(2, stride=2),
        nn.Flatten(),
        nn.Linear(400, 120),
        nn.ReLU(),
        nn.Linear(120, 84),
        nn.ReLU(),
        nn.Linear(84, 10)
    )
    return model
```

RESUMO DO CÓDIGO

```python
# Método para realizar a validação da predição durante o treinamento
def validate(model, data):
    total = 0
    correct = 0
    for i, (images, labels) in enumerate(data):
        images = images.cuda()
        x = model(images)
        value, pred = torch.max(x,1)
        pred = pred.data.cpu()
        total += x.size(0)
        correct += torch.sum(pred == labels)
    return correct*100./total
```

RESUMO DO CÓDIGO

```python
# Método de Treinamento
def train(numb_epoch=3, lr=0.001, device=device):
    accuracies = []
    cnn = create_lenet().to(device)
    cec = nn.CrossEntropyLoss() # Função de perda
    optimizer = optim.Adam(cnn.parameters(), lr=lr) # Otimizador Adam
    max_accuracy = 0
    for epoch in range(numb_epoch):
        for i, (images, labels) in enumerate(train_dl):
            images = images.to(device)
            labels = labels.to(device)
            optimizer.zero_grad()
            pred = cnn(images)
            loss = cec(pred, labels)
            loss.backward()
            optimizer.step()
        accuracy = float(validate(cnn, val_dl))
        accuracies.append(accuracy)
        if accuracy > max_accuracy:
            best_model = copy.deepcopy(cnn)
            max_accuracy = accuracy
            print("Saving Best Model with Accuracy: ", accuracy)
        print('Epoch:', epoch+1, "Accuracy :", accuracy, '%')
    plt.plot(accuracies)
    return best_model
```

RESUMO DO CÓDIGO

```python
# Método para realizar a predição de uma imagem da web
def inference_web_image(path, model, device):
    r = requests.get(path)
    with BytesIO(r.content) as f:
        img = Image.open(f).convert(mode="L")
        img = img.resize((28, 28))
        x = (255 - np.expand_dims(np.array(img), -1))/255.
    with torch.no_grad():
        pred = model(torch.unsqueeze(T(x), axis=0).float().to(device))
        return F.softmax(pred, dim=-1).cpu().numpy(), x
```

```python
# Método para realizar a predição de uma imagem local
def inference_local_image(path, model, device):
    img = Image.open(path).convert(mode="L")
    img = img.resize((28, 28))
    x = (255 - np.expand_dims(np.array(img), -1))/255.
    with torch.no_grad():
        pred = model(torch.unsqueeze(T(x), axis=0).float().to(device))
        return F.softmax(pred, dim=-1).cpu().numpy(), x
```

```python
# Inferir com Imagem da Web
# path = "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSq2h_xaDtemhitxk1AhEyzc5mYQu17d3Qb9Q&s"
# pred, x = inference_web_image(path, lenet, device=device)

# Inferir com Imagem Local
path = "./Train/1/Captura de tela 2024-06-16 211935.png"
pred, x = inference_local_image(path, lenet, device=device)

plt.imshow(x.squeeze(-1), cmap="gray")
pred_idx = np.argmax(pred)
print(f"Predicted: {pred_idx}, Prob: {pred[0][pred_idx]*100} %")
# print(pred)
```