Aprendizagem Profunda Aplicada – 2023/2024

Assignment 2

Turma PL1

Rafael Fernandes
2018294169

Inês Morgado
2016226593

1 Introdução

1.1 Protocolo de Treinamento

2 MNIST dataset

Este dataset é composto por 10 classes, catalogadas da seguinta maneira:

1. Zero (*Zero*)
2. Um (*One*)
3. Dois (*Two*)
4. Três (*Three*)
5. Quatro (*Four*)
6. Cinco (*Five*)
7. Seis (*Six*)
8. Sete (*Seven*)
9. Oito (*Eight*)
10. Nove (*Nine*)

Cada clase é representada por 7 000 imagens RGB, em que cada uma tem a dimensão de 1x28x28 pixels.

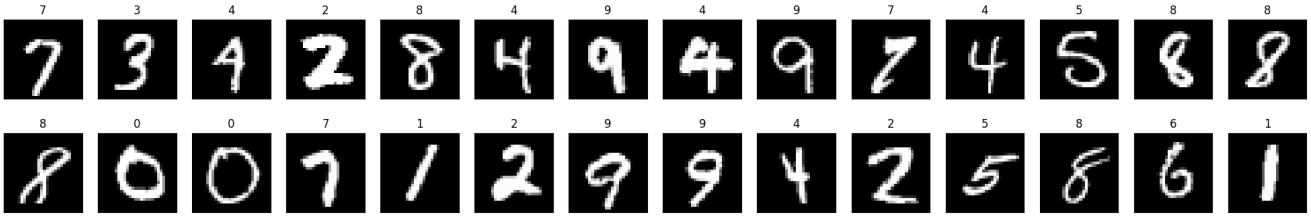

Figura 1: Dataset MNIST

2.1 Preparação do dataset

Este dataset está separado em um conjunto para ser usado para o treino, com 50 000 imagens, e as restantes, 10 000 imagens são para ser usadas no teste. Das imagens para o treino foram retiradas algumas imagens para depois serem usadas para a validação da rede.

Por fim o dataset ficou divididos nos seguintes conjuntos:

- Set de treino: 48 000 imagens;
- Set de validação: 12 000 imagens;
- Set de teste: 10 000 imagens

2.2 Autoencoder

```
(encoder_conv): Sequential(
  (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (1): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ELU(alpha=1.0)
  (3): Conv2d(8, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (5): ELU(alpha=1.0)
  (6): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (8): ELU(alpha=1.0)
  (9): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (11): ELU(alpha=1.0)
  (12): Conv2d(128, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (14): ELU(alpha=1.0)
)
(decoder_conv): Sequential(
  (0): ConvTranspose2d(16, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ELU(alpha=1.0)
  (3): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (5): ELU(alpha=1.0)
  (6): ConvTranspose2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (7): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (8): ELU(alpha=1.0)
  (9): ConvTranspose2d(32, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (10): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (11): ELU(alpha=1.0)
  (12): Upsample(scale_factor=2.0, mode='nearest')
  (13): ConvTranspose2d(8, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (14): BatchNorm2d(1, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (15): Sigmoid()
```

Rede do autoencoder

Figura 2: CNN

Nas *convolutional layers* ocorre a extração das features, que permitem fazer a distinção entre imagens de diferentes classes. Para isso é usado um filtro, kernel, que percorre todos os pixeis da imagem estimando o produto escalar entre os pesos do filtro e o valor do pixel da imagem. Devido a isto, nestas as camadas onde ocorre a maior carga computacional.

Neste trabalho foram utilizadas 5 *convolutional layers* com parâmetros de kernel=3, stride=1 e padding=1.

As *batch normalization layers* servem para reduzir o overfitting. Mesmo que tenhamos feito a normalização das imagens, data augmentation, pode acontecer que nas camadas intermédias da rede já não haja normalização, pois há grandes discrepâncias nos valores da média e desvio padrão.

O autoencoder é dividido em duas partes, o encoder e o decoder. O encoder serve para tira informação importante para se poder reduzir o tamanho da imagem e o decoder reconstroi a imagem a partir dessa mesma

informação. Para isso fez-se, o decoder quase como se fosse o espelho do encoder, mudando as camadas convolacionais para as suas transpostas.

2.2.1 Resultado

Como podemos verificar pelo outuput do decoder e pelo gráfico das perdas, o autoencoder reconstroi as imagens quase na perfeição.

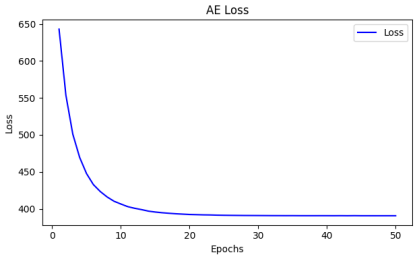

Figura 3: Gráfico das perdas ao longo do treino

2.3 Variational Autoencoder

O Variational Autoencoder (VAE) é um tipo de autoencoder que integra conceitos probabilísticos em sua arquitetura. Enquanto autoencoders tradicionais mapeiam dados para um único ponto no espaço latente, o VAE mapeia os dados para distribuições de probabilidade nesse espaço. Isso é alcançado introduzindo duas componentes adicionais, representando a média e o logaritmo da variância da distribuição latente. Durante o treinamento, o VAE utiliza a reparametrização, uma técnica que permite a geração de amostras estocásticas no espaço latente, viabilizando a geração de novos dados. A introdução de uma penalidade na forma de divergência de Kullback-Leibler na função de perda ajuda a regularizar o espaço latente, garantindo que ele se assemelhe a uma distribuição padrão.

A reparametrização é uma técnica fundamental no treinamento de Variational Autoencoders (VAEs) que permite a geração de amostras estocásticas no espaço latente de maneira diferenciável. Essa diferenciação é essencial para realizar retropropagação eficiente durante o treinamento. No contexto do VAE, a reparametrização é aplicada à amostragem da variável latente a partir da distribuição parametrizada pela média e logaritmo da variância.

Em vez de amostrar diretamente do espaço latente seguindo uma distribuição normal padrão, a reparametrização envolve a geração de amostras por meio de uma transformação determinística das variáveis aleatórias, utilizando a média e a variância. Em termos mais simples, ao invés de amostrar z diretamente, é calculada uma amostra ϵ a partir de uma distribuição normal padrão, e essa amostra é então transformada para o espaço latente usando a média e desvio padrão calculados.

Matematicamente, a reparametrização pode ser expressa como:

$$z = \mu + \epsilon \cdot \sigma$$

onde μ é a média da distribuição latente, σ é o desvio padrão (calculado a partir do logaritmo da variância), e ϵ é uma amostra aleatória da distribuição normal padrão.

Essa técnica permite que a retropropagação do gradiente ocorra suavemente durante o treinamento, pois as operações são diferenciáveis, permitindo a otimização eficaz dos parâmetros do VAE. A reparametrização é crucial para a geração de novos dados durante a fase de inferência do VAE.

2.3.1 Resultado

2.4 Denoising Autoencoder

O Denoising Autoencoder (DAE) é projetado para lidar com dados ruidosos, sendo uma ferramenta eficaz para tarefas de remoção de ruído. Durante o treinamento, o DAE é alimentado com dados corrompidos intencionalmente e é incentivado a reconstruir as entradas originais. Essa abordagem permite que o DAE aprenda representações robustas que podem filtrar eficientemente informações indesejadas ou irrelevantes. Ao corromper as entradas, o DAE aprende a reconstruir os dados subjacentes, capturando assim padrões e características importantes que são resilientes ao ruído. Essa capacidade de lidar com dados imperfeitos faz do DAE uma escolha valiosa em cenários nos quais a qualidade dos dados pode ser comprometida.

2.4.1 Resultado

3 KITTI dataset

Figura 4: Exemplo de uma image do dataset KITTI

3.1 Dataset

O KITTI dataset tem no total 500 images captadas através de câmaras RGB equipadas num carro. Os labels têm as três seguintes classes:

1. Carro (*Car*)
2. Ciclista (*Cyclist*)
3. Pessoa (*Person*)

3.1.1 Preparação do Dataset

Foi pedido para treinarmos a rede só com imagens de carros. Para isso fizemos um código que vai ler cada um dos labels e ver se a imagem correspondente tem algum carro. Depois basta eliminar as imagem que não têm carros e separando as restantes em dois subgrupos, um para o treino e outro para a validação.

3.1.2 YOLOv5

Esta é quinta versão da rede YOLO que são usada para deteção de objetos e para classificação de imagens. Esta rede é caraterizada por ser rápida e por ter uma grande accurancy. Neste trabalho optamos por usar a versão mais simples desta rede, a yolo5s.

3.1.3 Observações

Devido a erros do programa nao se conseguiu acabar o código e a tirar conclusões.

4 Conclusão

]