Tópicos Especiais em Computação XV – DIM0494 – 2023.1
Autoencoder em *ImageNet*

Lucas V. Bazante [1]

[1]Departamento de Informática e Matemática Aplicada (DIMAp)
Universidade Federal do Rio Grande do Norte - Natal, RN - Brasil

lucasbazante1@gmail.com

Resumo. *No contexto de Redes Neurais Artificiais, existem diversas técnicas e algoritmos cujos objetivos são dos mais variados. As mais comuns são técnicas de Classificação e Regressão, que se enquadram na categoria de Aprendizado Supervisionado. Porém, como sabemos, existe outra categoria de algoritmos, chamada Aprendizado Não-Supervisionado, cujo objetivo consiste em encontrar padrões, agrupamentos e novas representações dos dados. O modelo apresentado no presente trabalho trata-se de um autoencoder, um tipo de Rede Neural Artificial que tem por objetivo aprender uma nova representação dos dados, enquadrando-se na categoria de Aprendizado Não-Supervisionado.*

1. Introdução

Aprendizado de Máquina (AM) é uma subárea emergente da Inteligência Artificial (IA) que vem ganhando cada vez mais notoriedade e presença, suas técnicas sendo empregadas cotidianamente para solução de tarefas onde é requerido que o computador ”aprenda”, isto é, estime uma solução a partir de um conjunto de dados. Com o avanço de poder computacional e disponibilidade de grandes volumes de dados, a área se desenvolveu muito nos últimos anos, ganhando espaço e sendo capaz de realizar tarefas cada vez mais complexas.

Surge, neste contexto, a área de Aprendizado Profundo (AP), um subcampo de AM que faz uso diversas Redes Neurais Artificiais (RNA) de várias camadas, com arquiteturas cada vez mais complexas e técnicas capazes de extrairem informações dos dados de uma maneira que não era possível realizar com AM clássico, definindo o estado-da-arte em diversas tarefas.

A exemplo de tais redes com várias camadas e suas aplicações, podemos citar uma Rede Neural Convolucional (RNC), uma RNA que realiza convoluções nas camadas internas, cujos pesos são treináveis, gerando novas informações para servir de entrada às próximas camadas, ideal para lidar com imagens, visto que convoluções são utilizadas amplamente em tarefas de Processamento de Imagens e Visão Computacional.

No presente trabalho está contida a implementação, treinamento e resultados de um *autoencoder*, uma RNC, com várias camadas, treinada para receber imagens como entrada e devolver uma reconstrução da imagem original, utilizando o *dataset ImageNet* [Deng et al. 2009] como treinamento.

2. Fundamentação Teórica

2.1. Aprendizado de Máquina

O Aprendizado de Máquina é uma área de pesquisa que se propõe a estudar algoritmos capazes de aprender automaticamente, ou seja, sem serem explicitamente programados [Simon 2013]. Uma grande vantagem desses algoritmos é que seu objetivo é otimizar um critério de desempenho usando dados como exemplo [Alpaydin 2014].

Em outras palavras, pode ser definido como um processo onde o objetivo é otimizar um dado modelo, ou estimar uma função, a partir de um certo conjunto de dados X, conforme a constrição determinada por parâmetros particulares do modelo, chamados hiperparâmetros, que influenciam no aprendizado do algoritmo e são manualmente configurados antes do treinamento.

2.2. Aprendizado Não-Supervisionado

Comumente, modelos de AM tem por objetivo estimar uma função $f : X \to y$, onde X é o conjunto de dados e y é um conjunto de saídas esperadas, podendo ser este um conjunto finito de valores pré-definidos, ditos classes, ou uma variável contínua, ou seja, $y \subseteq \mathbb{R}$.

No caso de Aprendizado Não-Supervisionado, não há um conjunto de saídas desejadas para as instâncias de X, ou seja, não há classes nem valores contínuos a serem preditos. O modelo então aprende somente dos dados, sem indicações pré-definidas, descobrindo padrões ocultos nos dados.

A ideia principal é extrair informações significativas dos dados sem intervenção humana (dada pela definição de um valor esperado).

2.3. Redes Neurais Artificiais

Redes Neurais Artificiais são técnicas computacionais apresentando um modelo matemático inspirado na estrutura neural do cérebro.

O sistema nervoso é formado por um conjunto de células denominadas neurônios, que detém papel essencial no raciocínio humano. Estes são formados por dendritos e axônios, conjuntos de terminais de entrada e saída, respectivamente. A comunicação entre neurônios é realizada por sinapses, uma região onde dois neurônios entram em contato e transmitem impulsos nervosos. Conforme a intensidade do impulso, um dado neurônio pode ou não produzir uma substância neurotransmissora, transmitindo o impulso e inibindo ou excitando um próximo neurônio a este conectado por meio de uma sinapse.

Replicando matematicamente, cada neurônio se torna uma unidade de processamento, conectadas por canais de comunicação aos quais estão associados pesos. Uma coleção de neurônios e pesos forma uma camada, e as RNAs podem ter várias.

O funcionamento do modelo é resumido como se segue:

1. Os dados são apresentados à primeira camada, dita camada de entrada, que somente repassa à próxima; o número de neurônios corresponde ao número de atributos.
2. Cada neurônio apresentado é multiplicado pelo peso associado, que representa sua influência na saída da unidade, repassando o valor para um neurônio na próxima camada.

3. Para cada neurônio na próxima camada, é feita a soma ponderada dos neurônios da camada anterior, e esta soma alimenta uma função de ativação.
4. O resultado da função se torna a saída do neurônio que será repassada para a próxima camada de maneira similar.

O objetivo das RNAs consiste em aproximar uma função f, aprendendo melhores parâmetros que consistem numa melhor aproximação de f. O elemento chave por trás do aprendizado das RNAs está no *backpropagation*. O objetivo do *backpropagation* é ajustar os parâmetros internos de uma RNA. Calculando-se o gradiente de uma dada função de custo (associada à RNA previamente), considerando os pesos da rede. Os pesos são então atualizados de forma a se ajustarem aos dados de treinamento, começando da última camada, visando ajustá-los de forma a minimizar a função de custo atribuída, assim treinando a rede.

2.4. Aprendizado Profundo

Ao projetar atributos ou algoritmos para aprender atributos, o nosso objetivo é usualmente separar os fatores de variação que explicam os dados observados. Tais fatores de variação podem ser pensados como conceitos ou abstrações que nos ajudam a extrair sentido da rica variabilidade dos dados [Goodfellow et al. 2016]. Como exemplo, analisando imagens de flores, os fatores de variação incluem a cor das pétalas, seu formato, o brilho da imagem.

Os algoritmos de AM possuem a limitação de serem consideravelmente incapazes de aprender a distinguir os fatores de variação nos dados, pois a tarefa de extrair informações tão abstratas dos dados é extremamente difícil, pois representam níveis de abstração humanos.

Surge então o Aprendizado Profundo, um subcampo de AM que faz uso de RNAs com múltiplas camadas, onde cada camada tenta aprender uma representação simples dos dados, gerando uma hierarquia de conceitos, onde conceitos mais simples (camadas mais próximas da entrada) servem de apoio para o aprendizado de conceitos mais complexos. Isso é possível pois modelos compostos de diversas camadas podem aprender representações de dados com múltiplas camadas de abstração [LeCun et al. 2015].

Num exemplo com imagens, foco deste trabalho, uma camada seria capaz de detectar bordas, a camada seguinte detectaria contornos a partir das bordas, e a camada seguinte detectaria objetos a partir dos contornos.

2.5. Redes Neurais Convolucionais

Redes Neurais Convolucionais são estruturadas para processar dados que vem na forma de múltiplos *arrays* [LeCun et al. 2015]. O principal exemplo deste tipo de dado multidimensional são as imagens.

As RNCs possuem convoluções em suas camadas. A partir de um filtro, que é uma matriz de pesos, é aplicada uma convolução nos dados, onde cada área processada pelo filtro, dita campo receptor, por meio de uma soma ponderada, será um novo atributo na saída. Cada elemento da saída, chamada mapa de atributos, vem de um campo receptor específico. Cada mapa de atributos serve de entrada para a próxima camada (vários podem ser gerados).

Cada camada convolucional detectará um atributo dos dados, proporcionando assim a representação hierárquica de conceitos esperada de um modelo de AP.

Normalmente tais camadas são pareadas com uma camada de subamostragem, que simplifica a representação dos dados, proporcionando uma forma de representar os dados de maneira menos custosa. Ainda que o objetivo da camada convolucional seja detectar conjunções locais de atributos da camada anterior, a função da camada de subamostragem é juntar atributos semanticamente similares em um só [LeCun et al. 2015].

2.6. *Autoencoder*

Um *autoencoder* é uma técnica de Aprendizado Não-Supervisionado em que se aproveitam RNAs para a tarefa de Aprendizado de Representação. Especificamente, projeta-se uma RNA que força uma representação compacta dos conceitos da entrada original.

Um *autoencoder* consiste de duas partes: uma função de codificação $h = f(x)$ e um decodificador que produz a reconstrução $r = g(h)$ [Goodfellow et al. 2016]. É, então, uma composição das funções de codificação e decodificação, estimadas pela RNA, $(g \circ f)(x)$. A primeira parte da RNA codifica a entrada em uma representação menor, e a segunda parte reconstrói, a partir dessa representação, a saída, que tenta copiar a entrada original.

Se um *autoencoder* simplesmente aprende a atribuir $(g \circ f)(x) = x$, consiste simplesmente numa repetição da entrada, não sendo o objetivo. Alternativamente, *autoencoders* são projetados para serem incapazes de aprender a copiar perfeitamente a entrada, sendo usualmente restritos a copiarem aproximadamente, e copiar apenas entradas que sejam parecidas com os dados de treino [Goodfellow et al. 2016].

Pelo fato de que o modelo é forçado a priorizar quais aspectos da entrada deverão ser copiados, comumente aprende propriedades úteis do conjunto de dados [Goodfellow et al. 2016].

3. Metodologia

O objetivo do presente trabalho foi conceber, implementar e treinar um *autoencoder*, capaz de destruir e reconstruir imagens de maneira eficiente, e gerar, em sua arquitetura codificadora, vetores significativos.

3.1. Base de Dados e Pré-Processamento

A base de dados provém da base de imagens denominada *ImageNet*. Introduzida por [Deng et al. 2009], a base se propõe a ser uma ontologia de larga escala de imagens, dispostas hierarquicamente em classes diversas, possuindo milhões de exemplos classificados, sendo utilizada amplamente nas mais diversas tarefas de Visão Computacional e Aprendizado Profundo.

A versão mais comum desta imensa base é a disponibilizada pelo desafio *ImageNet Large Scale Visual Recognition Challenge* [Russakovsky et al. 2015], que avaliam algoritmos de detecção de objetos e classificação de imagens em larga escala. Este subconjunto dos dados contém 1,281,167 exemplos de treinamento e 50,000 exemplos de validação, todas as imagens sendo do tamanho fixo de 224×224, no formato *RGB*, ou seja, possuindo 3 canais de cor.

Considerando a complexidade da resolução, foi utilizada uma versão que contém as imagens redimensionadas para uma resolução menor, providas por

[Chrabaszcz et al. 2017], redimensionadas de maneira a perder o mínimo de informação possível. A resolução escolhida foi 32×32.

Ainda visando reduzir a complexidade e o tempo de treinamento, apenas 50% do conjunto de treinamento foi utilizado, contendo, ao fim, 640,584 imagens de treinamento.

Como pré-processamento, foi somente realizada a normalização *min-max* dos *pixels* das imagens, a fim de evitar alta dependência em valores altos de intensidade.

Os dados foram obtidos utilizando a *API TensorFlow Datasets*, uma coleção de bases de dados prontas para utlização.

3.2. Arquitetura do Modelo

O modelo *autoencoder* proposto é uma RNC de 20 camadas, com adição das camadas de entrada e saída. Há 16 camadas convolucionais, 2 camadas de subamostragem e 2 camadas de sobreamostragem. Cada uma das convoluções utiliza um filtro de tamanho 3×3, e as camadas de subamostragem e sobreamostragem possuem tamanho 2×2.

A inicialização dos filtros (modo como os pesos serão inicializados) das convoluções são todas *He Normal* [He et al. 2015] (exceto das camadas que geram a versão codificada e decodificada), gerando pesos uniformemente distribuídos conforme $\sqrt{\frac{2}{n}}$, onde n é o número de unidades do tensor de pesos. A inicialização foi escolhida devido a sua compatibilidade projetada com a função de ativação *ReLU*, que é a função escolhida para todas as convoluções não codificadoras e decodificadoras, a fim de introduzir linearidade nos dados.

Existem duas convoluções importantes a se notar: a codificadora e a decodificadora. Ambas utilizam a inicialização de filtro padrão, pois a função de ativação de ambas é a função *Sigmoid*. A escolha da função se dá ao fato de que, em cada uma dessas camadas convolucionais, tem-se por objetivo que a saída seja uma imagem, portanto cada campo receptor passará pela função *Sigmoide* a fim de que represente um valor de *pixel* normalizado entre 0 e 1.

Na Figura 1, pode-se ver a Arquitetura Codificadora em detalhe. Através dela a rede recebe as imagens como entrada. Similarmente, na Figura 2, pode-se ver a Arquitetura Decodificadora em detalhe. A junção de ambas as arquiteturas forma o *autoencoder* implementado neste trabalho.

Observando a figura, nota-se que as convoluções são utilizadas, juntamente com subamostragem e sobreamostragem, para alterar as dimensões das entradas, e forçar a rede a priorizar certos aspectos da imagem por meio do aprendizado, forçando uma reconstrução que não seja uma cópia cega da entrada, mas uma cópia aproximada baseada em propriedades aprendidas da imagem, como é a intenção de um *autoencoder*.

A parte decodificadora aprende quais aspectos manter, e gradativamente diminui as dimensões da imagem com as camadas de subamostragem, até por fim resultar numa codificação de tamanho $8 \times 8 \times 8$. A parte codificadora aprende os aspectos similarmente à decodificadora, porém gradativamente aumenta as dimensões da imagem por meio das camadas de sobreamostragem, até por fim resultar numa saída de tamanho $32 \times 32 \times 3$, justamente as dimensões da entrada.

O modelo foi implementado usando a biblioteca *Keras* [Chollet et al. 2015].

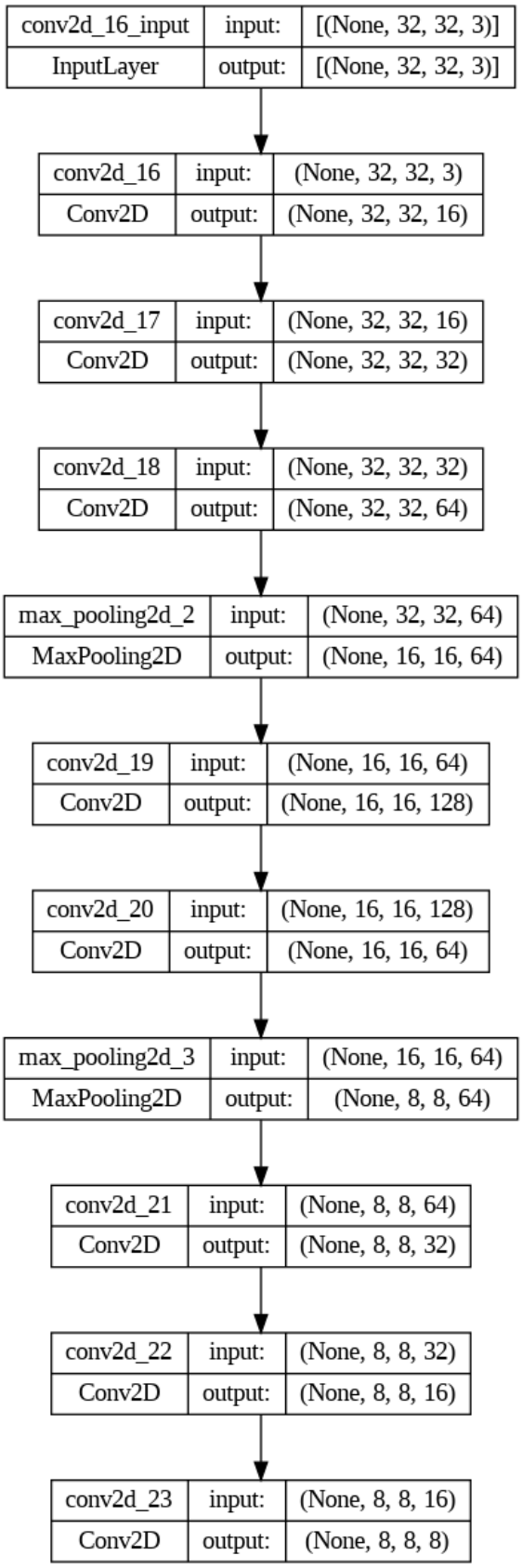

Figure 1. Arquitetura codificadora

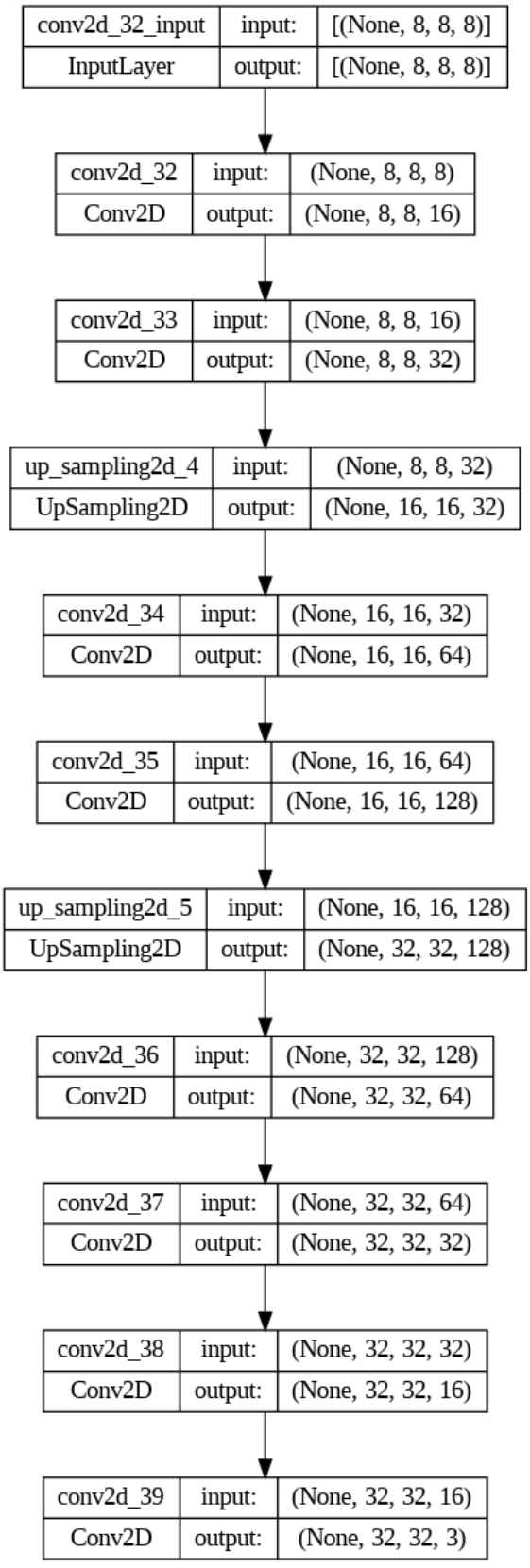

Figure 2. Arquitetura decodificadora

4. Resultados

O treinamento do modelo foi realizado em 30 épocas. O otimizador escolhido foi o *Adam*, e a função custo escolhida foi a Erro Quadrático Médio, ou, mais comumente em inglês, *Mean Squared Error* (MSE).

As imagens foram divididas em *batches* de 128 imagens cada, a fim de adequar o modelo aos dados de treinamento numa medida aceitável.

Tratando-se de um problema de Aprendizado Não-Supervisionado, as classes foram descartadas, e, como ”alvo”, vão as próprias imagens de entrada. Dessa forma, o MSE calculado será a diferença quadrática, *pixel* a *pixel*, da imagem original e da imagem reconstruída. O mesmo se aplica ao conjunto de validação.

A fim de prevenir épocas desnecessárias, um *callback* de *Early Stopping* foi adicionado, porém não chegou a ser ativado durante o treinamento, indicando melhora contínua do modelo.

O primeiro valor de MSE da validação foi registrado em 0.0050. Ao final do treinamento, este valor estava em 0.0020, apresentando melhora nos resultados. A Figura 3 mostra o gráfico contendo o histórico MSE do modelo ao longo das épocas, em treinamento e validação. O gráfico *appropriate fitting*, pois não há distância entre as curvas e nem indicação de incapacidade de aprendizado.

Cada época durou, em média, 3 minutos e 50 segundos. Ao todo, o treinamento foi realizado em aproximadamente 2 horas. O treinamento foi realizado na plataforma *Google Colaboratory*, com otimização de *GPU*.

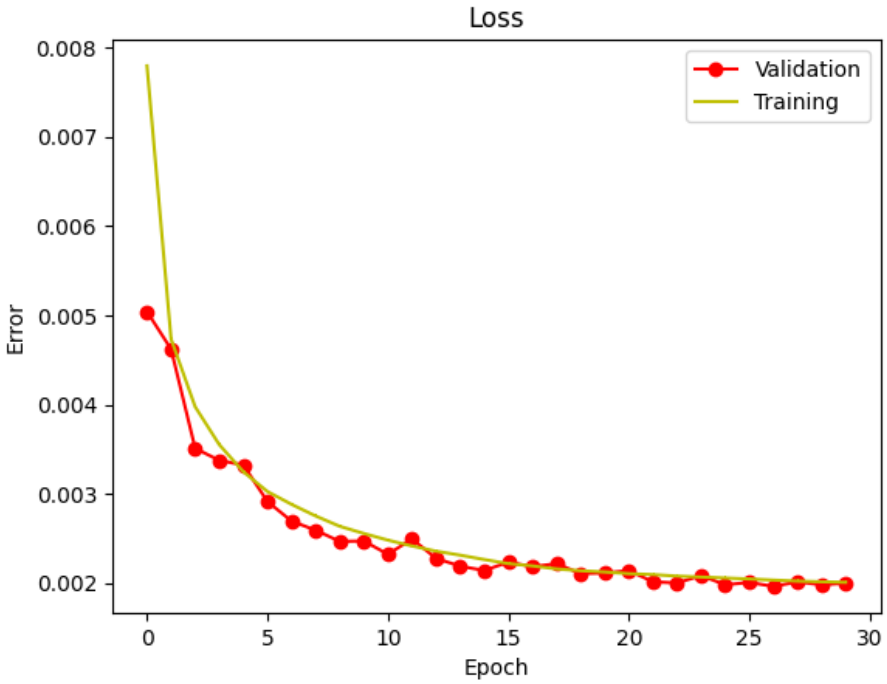

Figure 3. MSE ao longo das épocas

4.1. Demonstrações do modelo

A seguir vão algumas demonstrações do funcionamento do modelo já treinado. Cada figura contém duas linhas de imagens, sendo a primeira linha as entradas originais e a segunda linha as imagens reconstruídas pelo modelo.

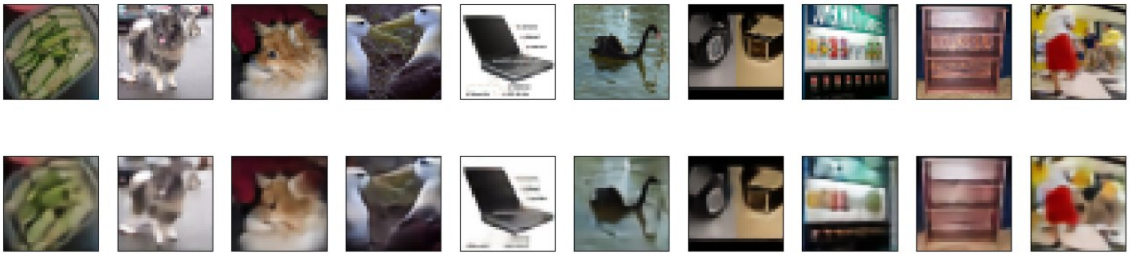

Figure 4. Demonstração 1

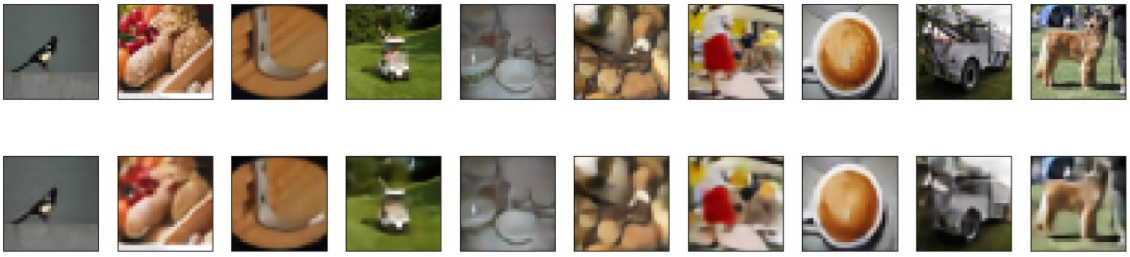

Figure 5. Demonstração 2

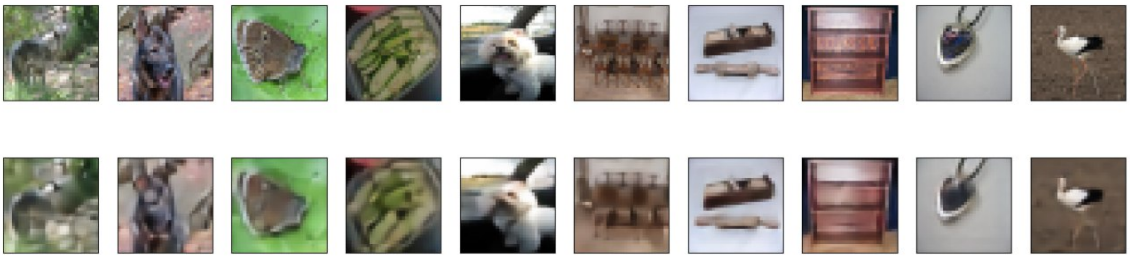

Figure 6. Demonstração 3

Figure 7. Demonstração 4

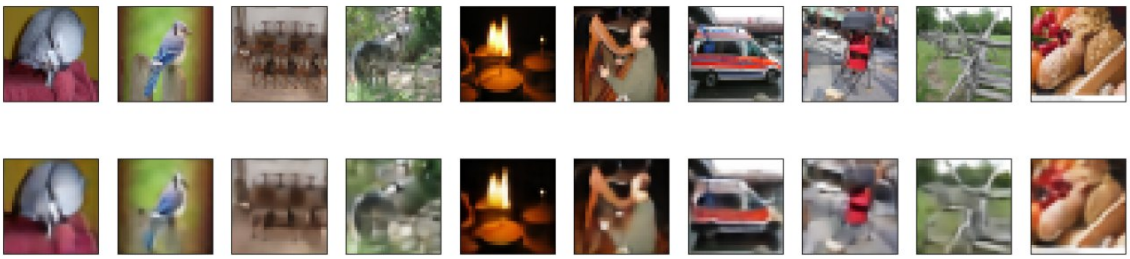

Figure 8. Demonstração 5

Pode-se notar que as imagens reconstruídas se assemelham a versões ”borradas” das originais, apresentando a mesma informação, porém codificadas de uma maneira diferente, como era esperado do modelo.

A seguir, veremos demonstrações da parte codificadora do modelo, que foram feitas da seguinte maneira:

1. Randomicamente se selecionam imagens do conjunto de dados.
2. Utilizando-se somente o modelo codificador, o conjunto de imagens é refatorado para conter codificações de tamanho $8 \times 8 \times 8$, conforme a arquitetura codificadora é capaz de fazer.
3. Achatam-se as codificações, a fim de obter vetores de 512 dimensões.
4. Com os vetores representando cada imagem, treina-se um modelo *k-Nearest Neighbors* (kNN), treinado com a métrica de distância similaridade do cosseno.
5. Com o kNN, são recuperados os vizinhos mais próximos de uma dada imagem de entrada, isto é, imagens similares a esta, conforme o critério estabelecido pelos vetores gerados pelo codificador achatado.

Cada figura contém uma linha de imagens, e, ao topo, o escore de similaridade atribuído à cada imagem em relação à imagem de entrada, conforme predito pelo modelo kNN com 6 vizinhos:

Figure 9. Demonstração 6

Figure 10. Demonstração 7

Figure 11. Demonstração 8

Figure 12. Demonstração 9

Figure 13. Demonstração 10

Figure 14. Demonstração 11

Figure 15. Demonstração 12

Figure 16. Demonstração 13

Notemos primeiramente que a imagem mais similar é sempre a imagem de entrada, indicando consistência na função de codificação.

Ao observar as demonstrações apresentadas, percebe-se que houve relativo sucesso em treinar a função codificadora de forma que retorne saídas significativas, priorizando certos atributos em detrimento de outros. Como era esperado, um dos atributos significativos é justamente a cor da imagem, pois os resultados com o kNN revelam que as imagens mais similares tem intensidades similares.

Em algumas das demonstrações é possível ver imagens relativamente discrepantes, mas sempre com uma gama de cores similares. Isso pode ser explicado dada a baixa resolução das imagens, não restando muitas outras informações significativas a considerar além da intensidade dos *pixels*.

Ainda assim, este não é o único fator determinante, pois, como podemos ver, algumas demonstrações indicam que há similaridade para além da intensidade no que foi codificado pelo modelo, notavelmente nas figuras 16, 15, 14 e 9.

5. Conclusão

Conclue-se que houve sucesso na implementação e treinamento do *autoencoder*, este sendo capaz de corretamente codificar e decodificar as entradas.

Em se tratando da reconstrução, o modelo produz versões aproximadas das imagens originais, não sendo somente cópias irrefletidas, mas sim resultado de uma seleção de atributos realizadas pelas camadas convolucionais sem perda de informação, como era esperado.

Com a parte codificadora, pode-se ver pela demonstração de imagens similares que houve sucesso em codificar atributos importantes das imagens de entrada, apesar da baixa resolução dos exemplos. As intensidades dos *pixels* tem grande influência na codificação das imagens, mas, mesmo assim, outras características também foram codificadas, como é possível ver nas demonstrações na seção anterior.

Tanto o resultado da MSE quanto as demonstrações visuais do *autoencoder* atestam o sucesso da implementação.

Para melhorias do modelo, pode-se utilizar uma diferente abordagem de reconstrução, projetando a rede de modo a manter certas características, como por exemplo gerar uma segmentação da imagem. Ainda mais, com o aumento da resolução das imagens, pode-se obter uma melhora na representatividade dos atributos selecionados pela parte codificadora.

6. Referências

Alpaydin, E. (2014). *Introduction to Machine Learning*. The MIT Press.

Chollet, F. et al. (2015). Keras.

Chrabaszcz, P., Loshchilov, I., and Hutter, F. (2017). A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

Simon, P. (2013). *Too Big to Ignore: The Business Case for Big Data*. Wiley Publishing, 1st edition.