Sistemas Autónomos - Aprendizagem não supervisionada utilizando autoencoders

Diogo F. Braga
University of Minho
Department of Informatics
4710-057 Braga, Portugal
Email: a82547@alunos.uminho.pt

Resumo. *As ações processadas pela visão humana têm sido transferidas para a área da Informática no sentido de potenciar a sua evolução. Este processamento é realizado informaticamente através da Visão Computacional, sendo o principal objetivo destes problemas usar os dados de qualquer imagem observada para inferir algo sobre o mundo. No entanto, nem sempre é fácil classificar os dados, pelo que, os autoencoders surgem como uma possível solução.*

1 Introdução

A classificação de imagens é um processo de extração e interpretação de informação a partir de dados digitais. Este processo inclui-se na área científico-tecnológica da Visão Computacional, uma das principais sub-áreas da Inteligência Artificial. O principal objetivo deste processo é emular a visão humana e, assim, ter capacidade de interpretar imagens. No entanto, nem sempre é fácil nem acessível realizar *labeling nos dados*, pelo que, os autoencoders surgem como uma possível solução.

Os autoencoders são um tipo específico de redes neuronais *feedforward*, onde a entrada é a mesma que a saída. Estes compactam a entrada num código de menor dimensão e, em seguida, reconstroem a saída dessa representação. O código é um "resumo" ou "compactação" da entrada, também chamada de representação do *latent space*. Um autoencoder consiste em 3 componentes: codificador, código e descodificador. Para construir um autoencoder, são necessárias três coisas: um método de codificação, um método de descodificação e uma função de perda para comparar a saída com o destino. Os autoencoders são principalmente um algoritmo de redução de dimensionalidade (ou compactação) com algumas propriedades importantes: *data-specific; lossy; unsupervised*. Este fenómeno acontece bastantes vezes em dados sensorizados, porque estes são apenas dados sem *labels*. Define-se, por isso, como um processo não supervisionado.

A demonstração é realizada sobre o *Mnist Fashion*, um *dataset* na escala do preto e branco. Para cada uma das fases são apresentados as arquiteturas utilizadas e os resultados obtidos. Estas fases serão todas testadas igualmente com 50 épocas, de forma a tornar a comparação justa. Durante o documento, a métrica de comparação tida em conta será a *loss*, significando assim que quanto mais acertar, melhor será considerado o modelo. A *accuracy* neste processo não supervisionado não se torna viável pois esta compara as imagens pixel a pixel, e tal não vai de encontro ao propósito final do modelo. No casos das imagens do *Mnist Fashion*, o modelo bastava acertar nos pixeis a preto para a *accuracy* possuir um valor elevado, o que não é suposto.

2 Autoenconder simples

Iniciando com uma arquitetura mais simplista, de forma a ter noção da capacidade dum autoencoder neste *dataset*, construiu-se a seguinte rede descrita na figura 1. Nesta arquitetura é importante fazer referência à camada intermédia, que define o *latent space*. Esta característica apresenta-se como fulcral no autoencoder, pois é um filtro que define que o autoencoder apenas compacta as *features* com maior relevância para as quais existe espaço, neste caso 32 *features*.

```
############## ENCODER ################
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                25120
=================================================================
Total params: 25,120
Trainable params: 25,120
Non-trainable params: 0
_________________________________________________________________

############## ENCODER + DECODER ################
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                25120
_________________________________________________________________
dense_2 (Dense)              (None, 784)               25872
=================================================================
Total params: 50,992
Trainable params: 50,992
Non-trainable params: 0
_________________________________________________________________
```

Fig. 1. Arquitetura associada ao autoencoder simples

Como a métrica mais importante neste método não supervisionado é a *loss*, esta é apresentada na figura 2. Nesta é possível visualizar a sua contínua descida, o que mostra que se encontra a aprender. Aliás, pelo facto de esta ainda não ter estabilizado, deviam ter sido realizadas mais algumas épocas.

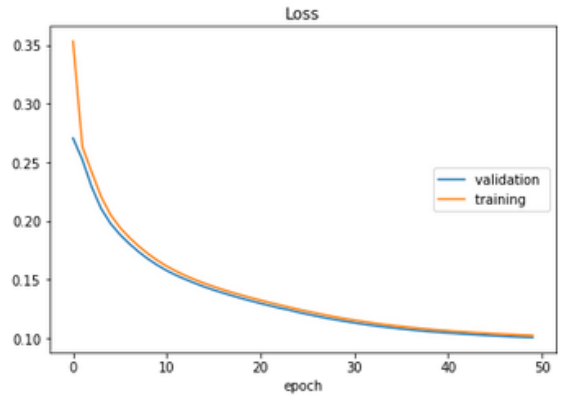

Fig. 2. Loss associado ao autoencoder simples

Com uma *loss* final fixada nos **0.10**, a identificação dos números do *dataset* foram na mesma todos bem identificados, tal como é visível na figura 3.

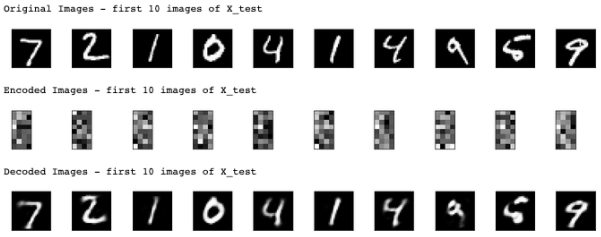

Fig. 3. Verificação do efeitos aplicado pelo autoencoder simples

3 Autoenconder deep

De forma a tentar obter melhores resultados, a arquitetura da rede foi evoluída, sendo acrescentadas mais duas camadas (figura 4).

Visualizando a *loss*, é possível constatar uma contínua descida, sinal da aprendizagem do autoencoder. Esta devia ter sido continuada pelo facto de ainda se encontrar em fase descendente, no entanto, no final de 50 épocas, atingiu um valor de *loss* de **0.084** (figura 5). Este valor é menor, o que indica que o aumento das camadas produziu efeitos, no entanto, estes não foram consideravelmente grandes, pelo que a verificação no *dataset* foi semelhante ao apresentado na figura 3.

```
############## ENCODER ################
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
=================================================================
Total params: 110,816
Trainable params: 110,816
Non-trainable params: 0
_________________________________________________________________

############## ENCODER + DECODER ################
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_5 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_6 (Dense)              (None, 784)               101136
=================================================================
Total params: 222,384
Trainable params: 222,384
Non-trainable params: 0
_________________________________________________________________
```

Fig. 4. Arquitetura associada ao autoencoder deep

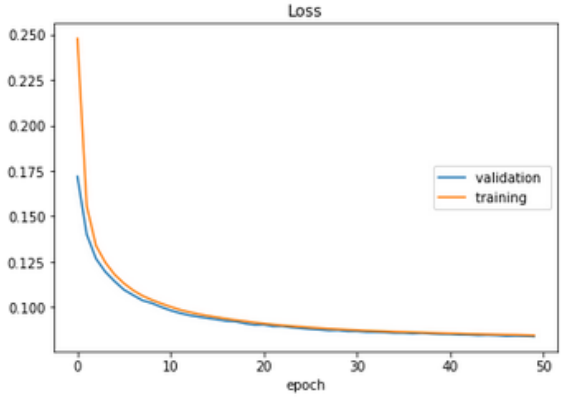

Fig. 5. Loss associado ao autoencoder deep

4 Autoenconder denoise

De forma a verificar o bom funcionamento do autoencoder, foi introduzido ruído nas imagens do *dataset*. A arquitetura utilizada foi a mesma da fase anterior, e os resultados de *loss* obtidos foram os apresentados na figura 6. Como seria expectável, este valor é um pouco mais elevado (**0.520**), pois os dados possuem interferência no seu normal estado e, consequentemente, a aprendizagem é afetada negativamente.

Apesar da introdução deste ruído, as imagens desenvolvidas pelo autoencoder corresponderam ao número ao qual estavam associadas, pelo que este se mostrou eficiente (figura 7). Este processo ocorre de uma forma positiva porque no *latent space* apenas ficam 32 *features*, e como essas são as 32 mais importantes, é possível descompactar a imagem com sucesso. Todas as restantes *features* com importância menor foram desconsideradas e, entre estas, o ruído foi esvaecido.

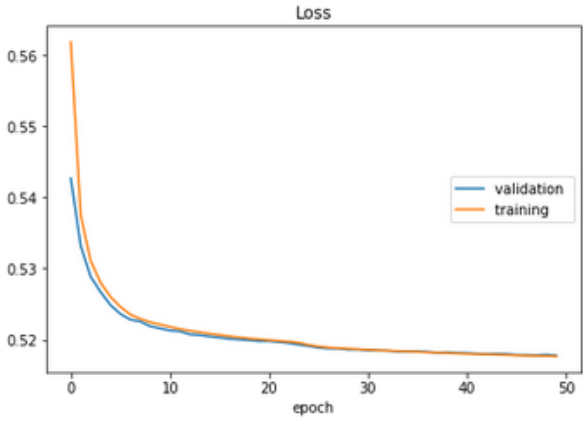

Fig. 6. Loss associado ao autoencoder denoise

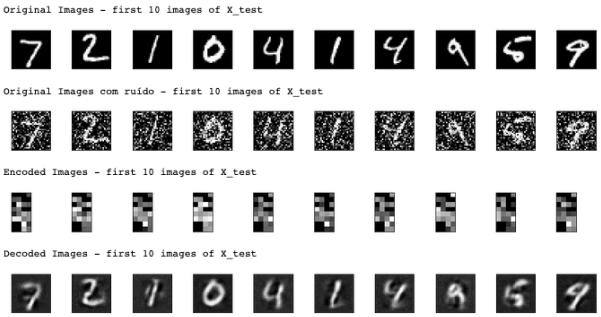

Fig. 7. Verificação do efeitos aplicado pelo autoencoder denoise

5 Variational autoencoder

Este é um caso de uso mais complexo de autoencoders, que aprende os parâmetros da distribuição de probabilidade modelando os dados de entrada, em vez de aprender uma função arbitrária.

Um VAE é um auto-codificador cuja distribuição de codificações é regularizada durante o treino para garantir que o *latent space* tenha boas propriedades, permitindo gerar novos dados. O termo "variacional" vem da estreita relação que existe entre a regularização e o método de inferência variacional nas estatísticas. Portanto, para poder usar o descodificador do autoencoder para fins generativos, é preciso ter certeza de que o *latent space* é suficientemente regular. Uma solução possível para obter essa regularidade é introduzir regularização explícita durante o processo de treino. Assim, um autoencoder variacional pode ser definido como um autoencoder cujo treino é regularizado para evitar *overfitting* e garantir que o *latent space* tem boas propriedades que possibilitem processos generativos. De forma a ter mais detalhe, neste modelo utilizam-se camadas convolucionais.

Assim como um autoencoder normal, um autoencoder variacional é uma arquitetura composta por um codificador e um descodificador, treinada para minimizar o erro de reconstrução entre os dados descodificados e os dados iniciais. No entanto, para introduzir alguma regularização do *latent space*, é procedida a uma ligeira modificação do processo de codificação/descodificação: em vez de codificar uma entrada como um único ponto, codificamos como uma distribuição no *latent space*. Os valores utilizados são a média e a variância do valor. São esses 2 vetores que são criados no *latent space*, como é possível visualizar na figura 8, nas camadas *dense2* e *dense3*.

Nestes modelos o número de épocas realizadas foi diminuído, devido à maior extensão de tempo necessário para o seu treino. A *loss* dos dados de treino (**148.89**), visível na figura 9, possui uma descida constante, o que leva a constatar que o treino devia continuar de modo a atingir melhores resultados. Neste caso, analisa-se a *loss* dos dados de treino apenas porque, com este modelo, a ideia é realizar ajuste sobre os próprios dados de treino, visto se tratar de um modelo não supervisionado.

```
############### ENCODER #################
Model: "encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 28, 28, 1)         0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 32)        320       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        18496     
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 64)        36928     
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 64)        36928     
_________________________________________________________________
flatten_1 (Flatten)          (None, 12544)             0         
_________________________________________________________________
dense_1 (Dense)              (None, 32)                401440    
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 66        
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 66        
=================================================================
Total params: 494,244
Trainable params: 494,244
Non-trainable params: 0
_________________________________________________________________

############### DECODER #################
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         (None, 2)                 0         
_________________________________________________________________
dense_4 (Dense)              (None, 12544)             37632     
_________________________________________________________________
reshape_1 (Reshape)          (None, 14, 14, 64)        0         
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 28, 28, 32)        18464     
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 28, 1)         289       
=================================================================
Total params: 56,385
Trainable params: 56,385
Non-trainable params: 0
_________________________________________________________________
```

Fig. 8. Arquitetura associada ao autoencoder variacional

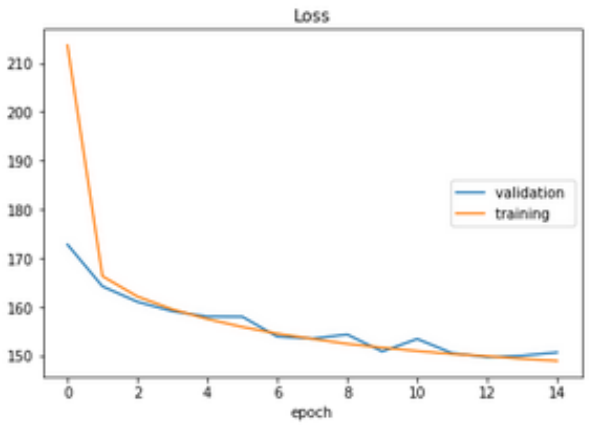

Fig. 9. Loss associado ao autoencoder variacional

6 Visualização de clusters Mnist

Como o *latent space* é bidimensional, existem algumas visualizações interessantes que podem ser realizadas, como observar os clusters das diferentes classes no plano 2D. Cada um desses clusters coloridos é um tipo de dígito, sendo que clusters próximos são dígitos estruturalmente semelhantes, visto que partilham informações no *latent space*.

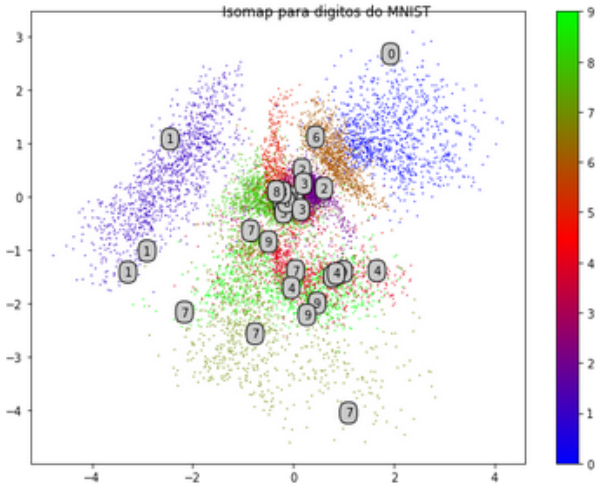

Fig. 10. Mapa com clusters associados a cada classe

Como o VAE é um modelo generativo, também é possível usá-lo para gerar novos dígitos. Na figura 11, digitaliza-se o plano latente, mostrando pontos latentes em intervalos regulares e gerando o dígito correspondente para cada um desses pontos. Este plano dá uma visualização do coletor latente que ”gera” os dígitos do *Mnist*.

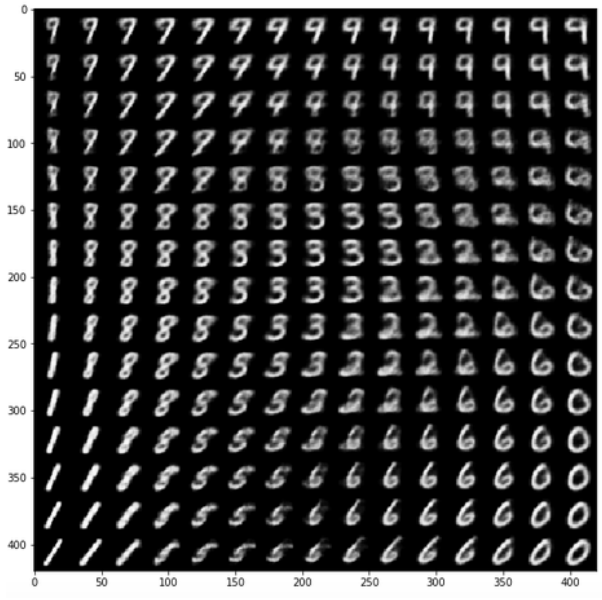

Fig. 11. Plano latente associado ao autoencoder variacional

De forma de testar a criação dos clusters deste modelo, visualizou-se o dígito gerado para uma das posições da representação no espaço, neste caso, a posição (3,2). O resultado foi o visualizado na figura 12, em que se verifica a criação do digito zero, como era suposto.

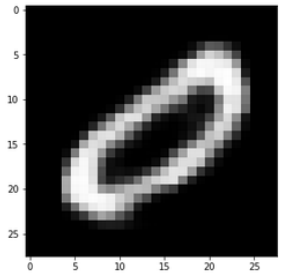

Fig. 12. Dígito gerado para as coordenadas (2,3)

7 Visualização de clusters Fashion Mnist

O mesmo processo realizado no *Mnist* foi também aplicado ao *Fashion Mnist*, *dataset* com maior complexidade. Utilizando a mesma arquitetura, o valor da *loss* esperava-se mais alta, e tal aconteceu, com valores de **264.35** ao fim de 15 épocas (figura 13). Este processo, mais uma vez, deveria ter sido continuado, dada a contínua curva descendente da *loss* nos dados de treino.

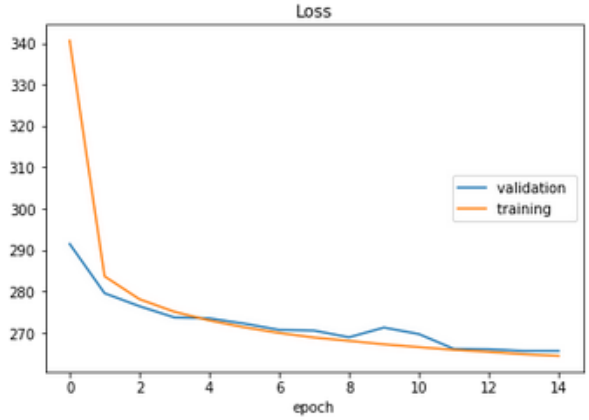

Fig. 13. Loss associado ao autoencoder variacional

Na figura 15 digitaliza-se o plano latente, mostrando pontos latentes em intervalos regulares e gerando a peça de roupa correspondente a cada um desses pontos. Este plano dá uma visualização do coletor latente que ”gera” as diferentes peças de roupa do *Fashion Mnist*.

De forma de testar a criação dos clusters deste modelo, visualizou-se duas peças de roupa geradas para duas posições da representação no espaço, neste caso, as posições (-3,2) e (0,-2). Os resultados foram os visualizados nas figuras 16 e 17, em que se verifica a criação de uma calças e de uns sapatos, respetivamente.

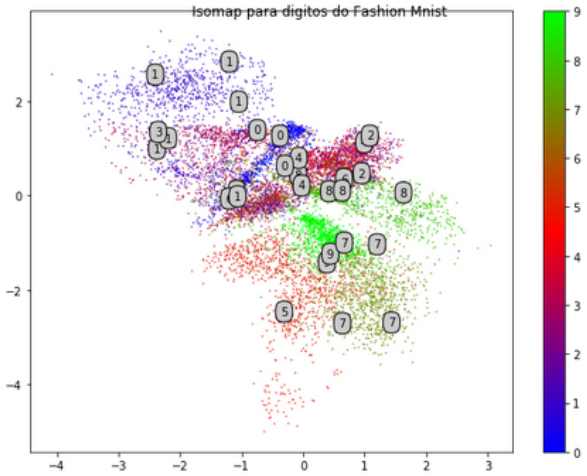

Fig. 14. Mapa com clusters associados a cada classe

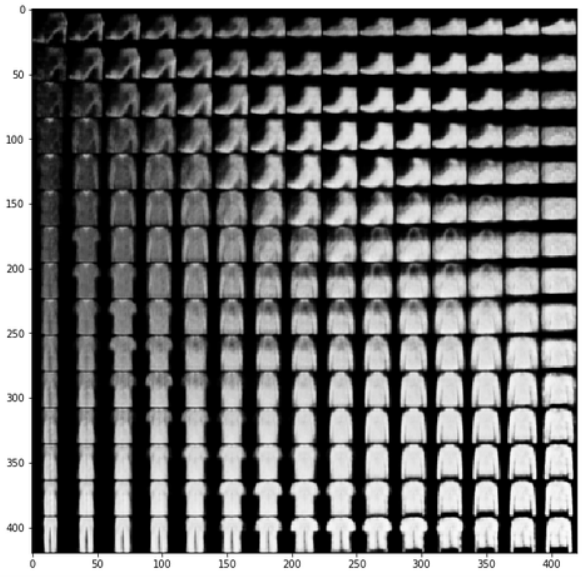

Fig. 15. Plano latente associado ao autoencoder variacional

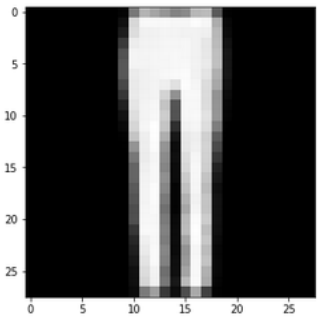

Fig. 16. Peça de roupa (calças) gerada para as coordenadas (-3,2)

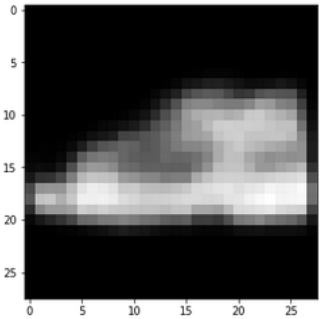

Fig. 17. Peça de roupa (sapato) gerada para as coordenadas (0,-2)

8 Conclusão

Deste documento e pesquisas relacionadas é importante, tal como no projetos anteriores, constatar a importância que o *hardware* possui na execução de camadas convolucionais, devido à grande quantidade de dados treinados e testados nas arquiteturas normalmente complexas neste tipo de rede. Este ponto foi uma limitação na realização de testes pois devido à falta de um GPU não foi possível aplicar outras bibliotecas com melhores rendimentos.

Após realização do projeto, constatou-se a importância que os autoencoders possuem para realização de redes não supervisionadas, ou seja, sem *labels* nos dados. De uma certa forma, o funcionamento base dos autoencoders é realizar um resumo das *features* mais importantes dum *dataset*, num género de compactação, e após isso disponibilizar esse modelo de forma a que seja aplicado em mais dados. O *latent space* é fulcral neste processo, pois é este que define o numero de *features* que são compactadas.

Deste projeto importante referir a *loss* dos dados de treino como a métrica tida em conta. Ao contrário da habitual utilização do *val_loss* nos modelos supervisionados, neste caso, como não existem *labels*, é realizado ajuste aos dados de treino, que por norma faz sentido ser a totalidade dos dados.

Os autoencoders variacionais, a outra variante de autoencoders abordada no projeto, apresentam uma pequena variação na utilização do *latent space*, calculando a média e a variância de cada valor e, após um *sampling*, é obtido um novo valor. É considerado um modelo generativo que utiliza camadas convolucionais para interpretar as imagens introduzidas como *input*.

Nos VAEs em específico, foi realizada uma pequena comparação entre o *Mnist* e o *Fashion Mnist*, e concluiu-se que, com a mesma arquitetura, foi possível obter uma solução aceitável em ambos os *datasets*. No entanto, dada a maior complexidade do *Fashion Mnist* e visualizando algumas zonas com mais clusters, a identificação das peças de roupa podia tornar-se um problema. Para isso, podia ser desenvolvida uma arquitetura também mais complexa.

Por fim, notar que os autoencoders têm bastantes aplicações no mundo real, como a deteção de anomalias, remoção de ruído, geração de imagens, etc. Devido à sua grande aplicabilidade, os autoencoders são um método não supervisionado com muitas vantagens e bastante utilizados.