

**Universidade de Coimbra**  
**Departamento de Engenharia Informática**



**Aprendizagem Probabilística e Reconhecimento de Padrões**

Relatório Projeto

**Mafalda Duarte - 2021236492**

**Rodrigo Santos - 2021236556**

**2023/2024**

## Parte I - Implementação do modelo Bayesiano

Para construirmos o modelo solicitado começamos por ir buscar os dados ao Kaggle tal como pedido no enunciado. Em seguida fomos fazer uma análise e tratamento dos dados. Começamos por verificar os valores em falta e concluímos que existiam um total de 126 linhas com valores em falta, o que representa aproximadamente 1,23% do dataset. As colunas que continham valores em falta eram ‘genre’ e ‘crew’. Estas colunas continham informação acerca do(s) género(s) do filme e da equipa de atores como tal decidimos apenas apagar as linhas com valores em falta visto que não era uma percentagem muito significativa e achamos que não faria sentido substituir estes pela moda dos valores.

No passo seguinte fomos eliminar algumas das colunas que pensamos não serem úteis para a construção do nosso modelo, particularmente: ‘names’, ‘overview’, ‘crew’, ‘orig\_title’ e ‘country’. As colunas ‘names’ e ‘orig\_title’ eram referentes ao título do filme o que não seria útil para a nossa análise, a ‘crew’ poderia ser interessante de utilizar pois os atores que participam nos filmes são muitas vezes importantes para o sucesso do mesmo, no entanto, devido à grande diversidade de valores e à complexidade da coluna decidimos deixá-la de parte tal como à coluna ‘overview’, por último a coluna ‘country’ não foi utilizada visto que achamos não a achamos particularmente importante.

Depois de ficarmos com as colunas desejadas decidimos retirar os filmes que ainda não tinham sido lançados, ou seja os filmes que na coluna ‘status’ tinham um valor diferente de ‘Released’, isto porque não fazia sentido para nós avaliar colunas como score ou revenue para os filmes que ainda não foram vistos.

Seguidamente focámo-nos nas colunas ‘date\_x’, ‘genre’, ‘score’, ‘budget’, ‘revenue’ e ‘orig\_lan’. Começamos pela coluna relativa à data de lançamento dos filmes, ‘date\_x’, que consideramos importante visto que há épocas do ano em que as pessoas têm mais disponibilidade para ir ao cinema e como tal pode afetar o sucesso do filme. Em vez de mantermos as datas específicas decidimos agrupar os valores em início, meio e fim do ano, de maneira a tornar a análise futura mais simples e geral. Quanto à coluna ‘genre’ optamos por transformá-la em vetores one-hot, apesar de fazer o nosso dataset aumentar o número de colunas achamos que seria importante pois há géneros de filmes que são muito mais apreciados do que outros e um filme ter mais do que um género pode também afetar a sua popularidade, avaliação etc. Relativamente às colunas ‘score’, ‘budget’ e ‘revenue’ definimos thresholds com base nos seus quartis e agrupamos os valores em ‘High’, ‘Medium’ e ‘Low’. Por último fomos à coluna ‘orig\_lan’ e começamos por ver como estavam distribuídos os valores. Depois de analisarmos então os valores decidimos mudar os valores para English ou Non-English. Isto porque, como podemos observar pelo gráfico apresentado no código, mais de metade dos valores são representados pela língua inglesa e então não nos pareceu essencial fazer distinção entre as línguas mais pequenas.

Demos assim por finalizado o nosso pré-processamento e passamos então à criação do nosso baysian model. Nesta etapa tentamos criar primeiramente um modelo seguindo apenas a nossa lógica e colocamos as dependências que nos pareceram mais óbvias e importantes:

- ('date\_x', 'revenue\_cat')
- ('orig\_lang', 'revenue\_cat')
- ('Crime', 'revenue\_cat')
- ('Documentary', 'revenue\_cat')
- ('Comedy', 'revenue\_cat')
- ('War', 'revenue\_cat')
- ('Mystery', 'revenue\_cat')
- ('Romance', 'revenue\_cat')
- ('Action', 'revenue\_cat')
- ('Music', 'revenue\_cat')
- ('Western', 'revenue\_cat')
- ('Science Fiction', 'revenue\_cat')
- ('Adventure', 'revenue\_cat')
- ('Horror', 'revenue\_cat')
- ('Drama', 'revenue\_cat')
- ('Animation', 'revenue\_cat')
- ('Fantasy', 'revenue\_cat')
- ('TV Movie', 'revenue\_cat')
- ('Family', 'revenue\_cat')
- ('History', 'revenue\_cat')
- ('Thriller', 'revenue\_cat')
- ('score\_cat', 'revenue\_cat')
- ('budget\_cat', 'revenue\_cat')
- ('Crime', 'budget\_cat')
- ('Documentary', 'budget\_cat')
- ('Comedy', 'budget\_cat')
- ('War', 'budget\_cat')
- ('Mystery', 'budget\_cat')
- ('Romance', 'budget\_cat')
- ('Action', 'budget\_cat')
- ('Music', 'budget\_cat')
- ('Western', 'budget\_cat')
- ('Science Fiction', 'budget\_cat')
- ('Adventure', 'budget\_cat')
- ('Horror', 'budget\_cat')
- ('Drama', 'budget\_cat')
- ('Animation', 'budget\_cat')
- ('Fantasy', 'budget\_cat')

- ('TV Movie', 'budget\_cat')
- ('Family', 'budget\_cat')
- ('History', 'budget\_cat')
- ('Thriller', 'budget\_cat')

No entanto, por motivos computacionais, não conseguimos obter as tabelas cpd e por isso fomos então usar um modelo que cria automaticamente as dependências. Decidimos utilizar o algoritmo HillClimbSearch e experimentar 3 *scoring\_methods* : o K2Score, o BDeuScore e o BicScore.

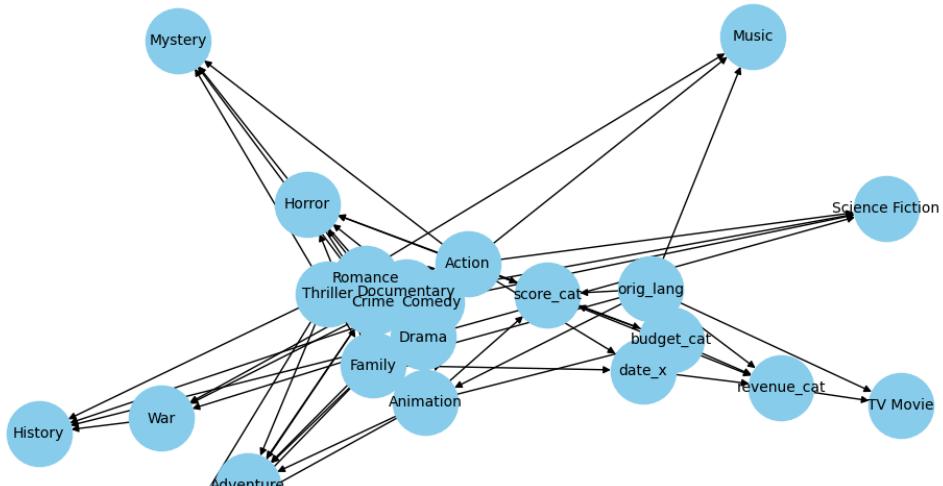
Decidimos também fazer um modelo cujas arestas eram aquelas que estavam presentes em pelo menos 2 dos 3 modelos (*modelo\_*), um modelo cujas arestas fossem a interseção das arestas dos 3 modelos (*modelo\_inter*), um modelo em que realizamos algumas alterações no modelo bic original de acordo com a nossa lógica (*modelo\_bic\_mod*) e um modelo em que realizamos alterações no modelo K2 original de acordo com a nossa lógica (*modelo\_k2\_mod*). Depois, de forma a nos ajudar a escolher que modelo usar, fomos calcular os scores para cada um e os resultados obtidos foram os seguintes:

	BDeu	K2	BIC
modelo_k2	-153839.61	28234285.84	-376788039.74
modelo_bdeu	-101818.02	-101798.34	-105808.35
modelo_bic	-102356.77	-102363.26	-103036.57
modelo_	-106674.57	-106677.93	-106867.62
modelo_inter	-106258.93	-106213.55	-106428.86
modelo_bic_mod	-102699.94	-102650.01	-103463.35
modelo_k2_mod	-145009.27	7885775.68	-106354716.47

Como podemos observar, o modelo 1 e 7 têm valores muito altos de K2, o que pode indicar overfitting. Quanto aos restantes modelos apresentam valores bastante semelhantes, acabamos por escolher o *modelo\_bic\_mod* porque foi um dos modelos onde incluímos um pouco da nossa lógica.

Fomos então gerar as tabelas cpds e criar a imagem da nossa rede bayesiana.

Estrutura da Rede Bayesiana *modelo\_bic\_mod*



Analisando a nossa rede conseguimos perceber que não é ideal, na nossa opinião nós como '*revenue\_cat*' e '*budget\_cat*', por exemplo, deveriam ter mais ligações por serem variáveis que nos parecem depender muito de outras. Idealmente gostaríamos de juntar as dependências geradas pelo modelo àquelas que fizemos por lógica inicialmente mas isso não é possível. Percebemos também que caso fizéssemos o modelo apenas só pela nossa lógica não teríamos adicionado dependências entre, por exemplo, géneros de filmes que também são importantes. Ao fim da análise passamos ao passo final e fomos realizar algumas inferências. Começamos por uma simples, a probabilidade de um filme ser uma animação e depois passamos para algumas um pouco mais complexas como a probabilidade conjunta da *receita* e do *score* sabendo que o *budget* é pequeno. Depois realizamos mais algumas, que nos pareceram pertinentes, e observamos alguns valores que nos surpreenderam, por exemplo para as duas últimas inferências onde calculamos a probabilidade, não conjunta, da *receita* e do *budget* sabendo que a *orig\_lang* é não inglesa e o mesmo mas sabendo que a *orig\_lang* é inglesa. De repente a nossa ideia seria que o *budget* e a *receita* teriam uma probabilidade de ser altos em filmes em inglês, no entanto aconteceu o contrário talvez por termos muitos mais exemplos de filmes ingleses do que das outras línguas.

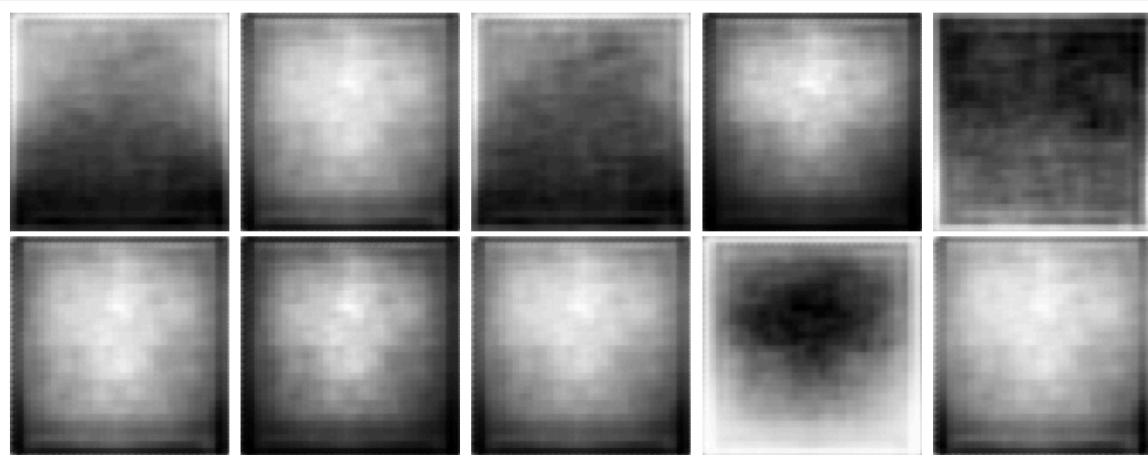
## Parte II - Implementação de um modelo baseado em VAE

O primeiro passo da segunda parte do nosso projeto foi redimensionar todos os posters para 128x128 e guardar as imagens redimensionadas com o nome do respetivo *movieId* para facilitar a associação dos dois datasets fornecidos. Em seguida, fomos buscar 1000 dos 62 061 posters para facilitar o processo de testar o nosso código.

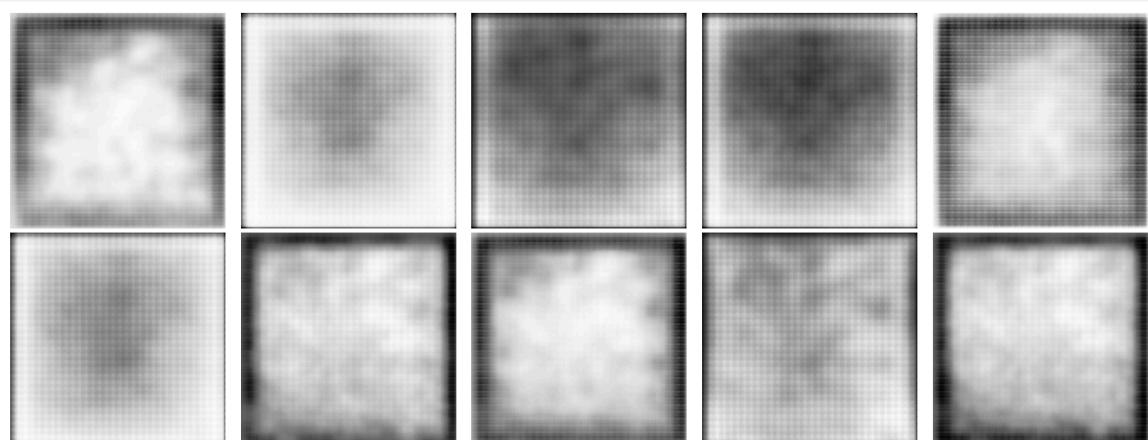
Começamos então por fazer o VAE, para isso utilizamos o código de um dos assignments que realizamos e fizemos algumas alterações, nomeadamente na função *load*. Esta função tem como objetivo carregar as imagens de um caminho *c* e processá-las para lidar com possíveis erros de maneira a que todas as imagens estejam no formato esperado e depois devolvê-las num array normalizado, acrescentamos também uma variável para guardar o nome dos ficheiros para utilizarmos no cVAE. Depois de fazermos as alterações na função *load* ajustamos também o parâmetro *input\_dim*. Optamos por realizar diversas experiências mudando um pouco os parâmetros.

Começamos por utilizar então 1000 posters:

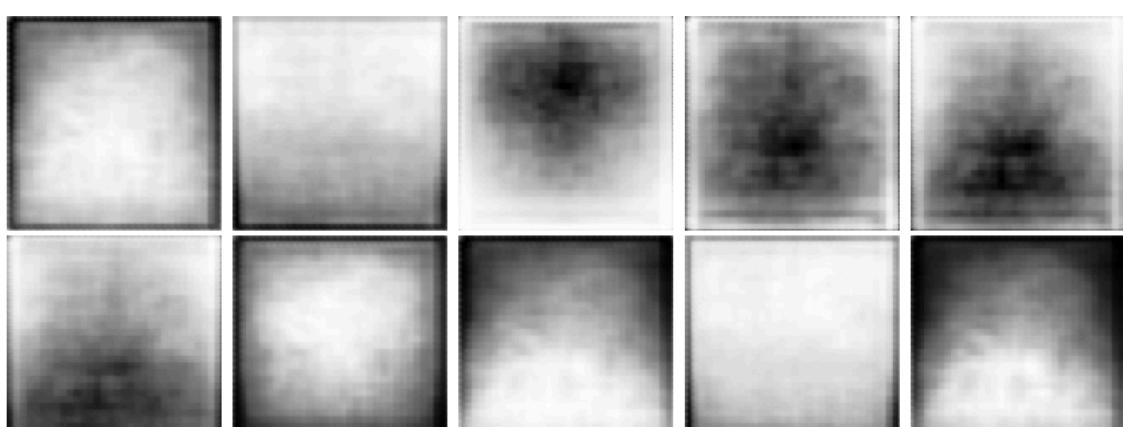
1. Modelo treinado com um *learning\_rate* de 0.0005, um *reconstruction\_loss\_factor* de 100, um *batch\_size* de 32 e número de *epochs* igual a 50. Output para 10 vetores gerados aleatoriamente:



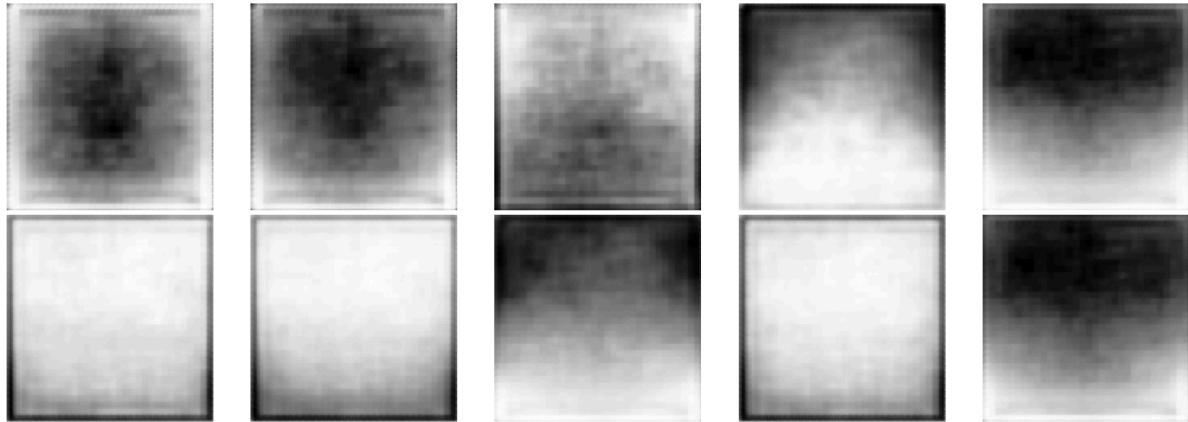
2. Modelo com um *learning\_rate* de 0.0001, um *reconstruction\_loss\_factor* de 100, um *batch\_size* de 64 e número de *epochs* igual a 100. Output para 10 vetores gerados aleatoriamente:



3. Modelo com um *learning\_rate* de 0.0005, um *reconstruction\_loss\_factor* de 300, um *batch\_size* de 32 e número de *epochs* igual a 50. Output para 10 vetores gerados aleatoriamente:



4. Modelo com um *learning\_rate* de 0.0005, um *reconstruction\_loss\_factor* de 1000, um *batch\_size* de 32 e número de *epochs* igual a 50. Output para 10 vetores gerados aleatoriamente:



Como podemos observar pelo resultados mostrados acima, não estávamos a conseguir obter bons outputs com as 1000 amostras e por isso decidimos então aumentar o número para 5001 e realizar novas experiências:

1. Modelo com um *learning\_rate* de 0.0005, um *reconstruction\_loss\_factor* de 900, um *batch\_size* de 32 e número de *epochs* igual a 25. Output para 10 vetores gerados aleatoriamente:



- Modelo com um *learning\_rate* de 0.0005, um *reconstruction\_loss\_factor* de 400, um *batch\_size* de 32 e número de *epochs* igual a 25. Output para 10 vetores gerados aleatoriamente:



- Modelo com um *learning\_rate* de 0.0005, um *reconstruction\_loss\_factor* de 100, um *batch\_size* de 64 e número de *epochs* igual a 25. Output para 10 vetores gerados aleatoriamente:



Foi notável a diferença de 1000 para 5001 posters. Conseguimos ver que o modelo reconstruiu um poster perceptível, no entanto foi sempre o mesmo o poster do filme ToyStory. Para tentarmos melhorar o modelo reduzimos o *reconstruction\_loss\_factor*. Na última experiência conseguimos ver que há uma diminuição da nitidez dos posters gerados em relação aos anteriores. Já da primeira para a segunda experiência não houve grande diferença.

Como ainda estávamos muito longe do número de posters total e já tínhamos o código a funcionar decidimos tentar treinar o modelo com todos, no entanto não iríamos conseguir por falta de tempo. Optamos então por treinar o modelo com 10 001 amostras e fazer de novo algumas experiências.

1. Modelo com um *learning\_rate* de 0.0005, um *reconstruction\_loss\_factor* de 2000, um *batch\_size* de 64 e número de *epochs* igual a 15. Output para 10 vetores gerados aleatoriamente:



2. Modelo com um *learning\_rate* de 0.0005, um *reconstruction\_loss\_factor* de 1000, um *batch\_size* de 32 e número de *epochs* igual a 25. Output para 10 vetores gerados aleatoriamente:



3. Modelo com um *learning\_rate* de 0.0005, um *reconstruction\_loss\_factor* de 800, um *batch\_size* de 64 e número de *epochs* igual a 15. Output para 10 vetores gerados aleatoriamente:



4. Modelo com um *learning\_rate* de 0.0005, um *reconstruction\_loss\_factor* de 500, um *batch\_size* de 32 e número de *epochs* igual a 25. Output para 10 vetores gerados aleatoriamente:



- Modelo com um *learning\_rate* de 0.0005, um *reconstruction\_loss\_factor* de 100, um *batch\_size* de 64 e número de *epochs* igual a 15. Output para 10 vetores gerados aleatoriamente:



- Modelo com um *learning\_rate* de 0.0005, um *reconstruction\_loss\_factor* de 90, um *batch\_size* de 32 e número de *epochs* igual a 15. Output para 10 vetores gerados aleatoriamente:



Ao treinarmos com as 10 001 amostras começamos por colocar um *reconstruction\_loss\_factor* com um valor muito grande e fomos diminuindo. Como podemos concluir pelas imagens acima com um *reconstruction\_loss\_factor* de 500 um *batch\_size* de 32 e número de *epochs* igual a 25 este modelo com mais amostras conseguiu gerar posters perceptíveis ao contrário do que acontece com os modelos de 1000 amostras que nem com um *reconstruction\_loss\_factor* de 1000 , um *batch\_size* de 32 e número de *epochs* igual a 50 nos deu bons resultados. Com *reconstruction\_loss\_factor* igual a 90 o output já foi claramente menos nítido do que nos outros.

## Parte II - Implementação de um modelo baseado em CVAE

Começamos por criar uma lista com o id dos posters que obtemos pelos filenames da função `load` e vamos realizar o one-hot encoding dos genres, visto que vamos utilizar estes valores para treinar o modelo. Tal como este one-hot encoding, também damos como input os posters dos respectivos filmes, pelo que fazemos a ordenação destes 2 parâmetros de entrada para cada filme e os seus géneros correspondam à imagem correta.

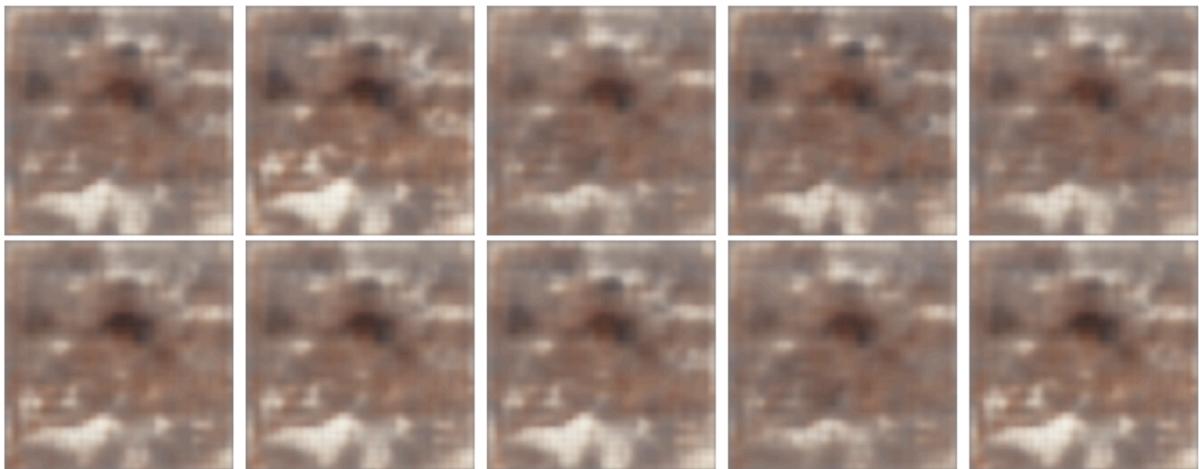
Em seguida, utilizamos a arquitetura do CVAE implementado num assignment, ajustando apenas `input_dim`. De maneira idêntica ao VAE, realizámos também várias experiências, alterando o número de posters, epochs e `batch_size`. Começamos por treinar o modelo com 1000 posters com um `learning_rate` de 0.0005, um `batch_size` de 32 e número de `epochs` igual a 50.

```
cvae.compile_model(learning_rate=0.0005)  
  
history = cvae.model.fit([posters, genres_one_hot], posters, epochs=50, batch_size=32)
```

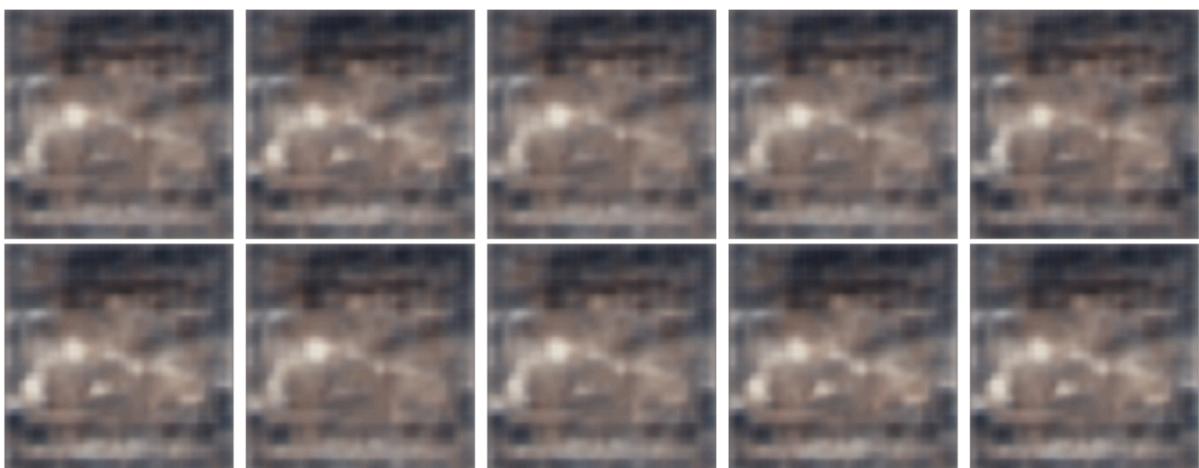
1. Geramos 10 posters para diferentes géneros (onde os géneros possíveis são: 'Drama', 'Film-Noir', 'Crime', 'Documentary', 'Children', 'Adventure', 'Romance', 'Action', 'Musical', 'Animation', 'IMAX', 'Fantasy', 'Mystery', 'Horror', 'Comedy', 'War', 'Sci-Fi', 'Thriller', 'Western'), sendo os resultados obtidos:



Genre: Action



Genre: Adventure



Genre: Animation

Como podemos ver, todos os posters gerados, independentemente do género, geraram todos o mesmo poster, sendo apenas visível ligeiras diferenças em alguns grupos de pixels, pelo que podemos tirar as conclusões que o problema ou está em overfitting ou no número reduzido de epochs para a quantidade de dados.

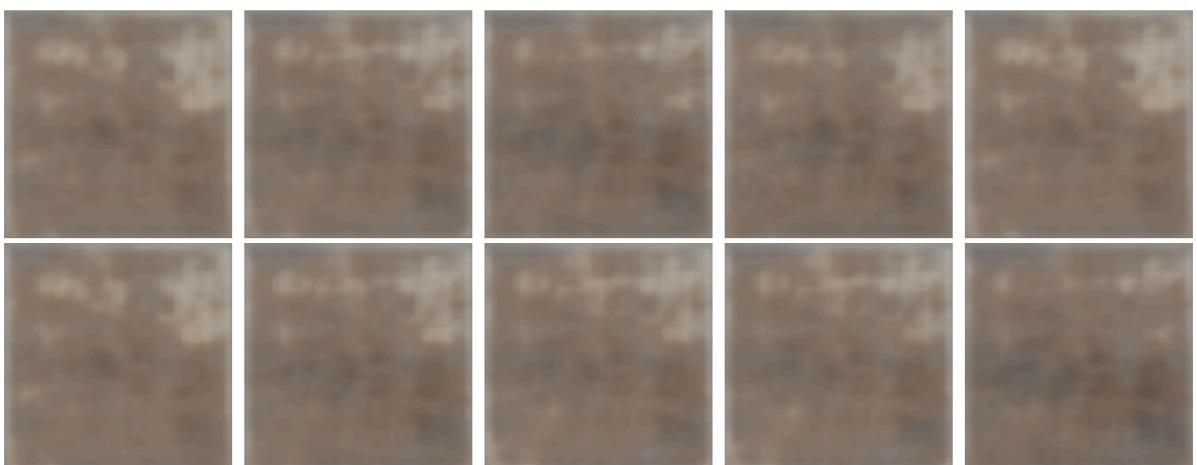
Na 2<sup>a</sup> experiência optámos por utilizar 5001 posters para o treino do modelo, com um *learning\_rate* de 0.0005, um *batch\_size* de 64 e número de *epochs* igual a 25.

```
cvae.compile_model(learning_rate=0.0005)  
history = cvae.model.fit([posters, genres_one_hot], posters, epochs=25, batch_size=64)
```

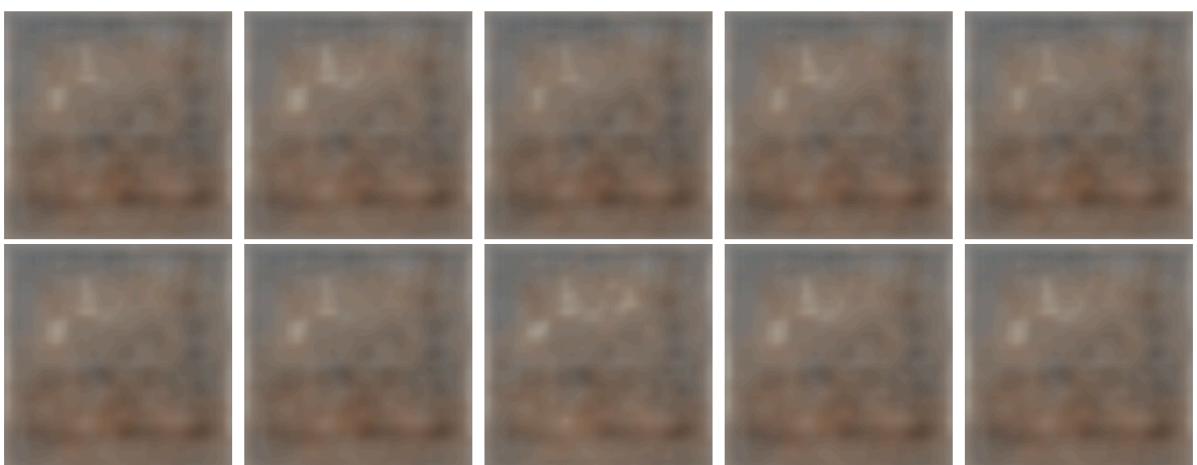
Geramos novamente 10 posters, obtendo os resultados:



Genre: Action



Genre: Adventure



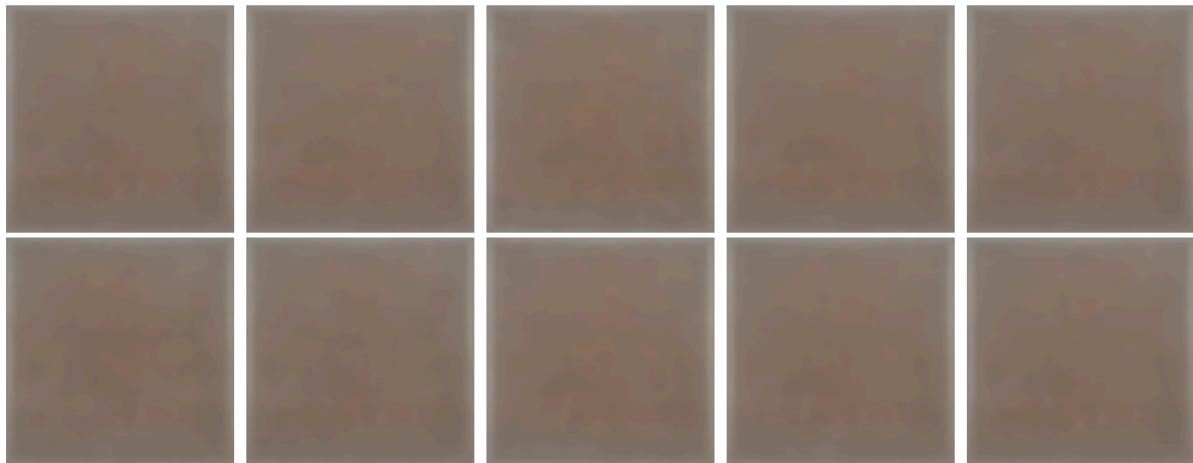
Genre: Animation

Em relação à experiência anterior continuamos com o mesmo problema de estar a gerar o mesmo poster de maneira bastante similar. Obtemos piores resultados do que anteriormente quando devíamos ter obtido resultados melhores, visto que utilizámos mais posters para o treino, sendo então o problema a redução de epochs e o aumento do batch\_size.

Na última experiência utilizamos 10001 posters, *learning\_rate* de 0.0005, um *batch\_size* de 64 e número de *epochs* igual a 10.

```
cvae.compile_model(learning_rate=0.0005)  
history = cvae.model.fit([posters, genres_one_hot], posters, epochs=10, batch_size=64)
```

Geramos igualmente 10 posters, obtendo os seguintes resultados:



Genre: Action



Genre: Adventure



Genre: Adventure

Como era de esperar, obtemos os piores resultados em que não há nada que seja perceptível, apesar de ser onde utilizámos um maior número de posters, foi também onde utilizámos o número mais reduzido de epochs.