Análise Aplicação Rede CNN ao problema Cifar-10

Previamente, todo o contacto estabelecido entre o autor e as redes convolucionais, focou-se na sua aplicação a problemas de teor unidimensional (utilização de convoluções 1D). Dessa forma e de modo a que fosse possível aumentar o “*know-how*”, foi aplicado o conceito de CNN também a problemas que exigissem a aplicação de convoluções 2D.

Para tal, foi considerada a aplicação deste tipo de redes a um típico problema de classificação de imagens. Aliás, este é considerado o âmbito mais abrangente, da sua aplicação. O problema escolhido, foi o conhecido *Cifar-10,* um dos problemas mais utilizados em contexto académico/aprendizagem. Este problema é também considerado uma *benchmark* de verificação da qualidade dos modelos aplicados, porque ao contrário de outros problemas como o MNIST, revela-se um problema mais complexo de resolver. Com isto, é expectável que o autor aumente o seu conhecimento quer na aplicação de redes CNN (aplicação de redes convolucionais 2D e ainda o estudo e aplicação de redes CNN já devidamente testadas e formuladas, como: *AlexNet* ou *VGGNet*), mas também na sua otimização.

De modo a reduzir a carga computacional e tempo exigido na execução e otimização dos modelos, foi considerada apenas a utilização de 4 das 10 classes do problema. As classes consideradas, foram: imagens de cães, gatos, cavalos e sapos. Foram escolhidas estas classes, ao invés, de outros devido ao facto de existir uma maior correlação entre estas (mesma família – animais), tornando assim o estudo mais realista.

Esta exigência levou a que fosse necessário, efetuar uma devida reformulação de todo o conjunto de dados. Isto é, é necessário limpar os vários conjuntos de dados, de treino e de teste, removendo assim as 6 classes do problema que não irão ser estudadas, definir código que permite a utilização de um nº predefinido de dados de treino e ainda de teste. Depois, foi ainda necessário reformular as *labels* de cada classe, de modo a que estivessem em conformidade com o *output* devolvido, da aplicação das redes convolucionais.

Finalmente, foi ainda necessário recorrer à aplicação de técnicas de transformação dos dados, como a aplicação de normalização aos dados do problema (treino e teste), de modo a que todos os *pixels* estejam definidos num espaço semelhante, permitindo assim que os gradientes calculados possam convergir mais rapidamente[1].

Depois, de aplicado todo este processo foi necessário proceder à aplicação de uma rede CNN que conseguisse dar resultados adequados à resolução deste problema. Decidi, efetuar um estudo da arte sobre algumas das redes mais utilizadas, em problemas de classificação de imagens (e não só, são vários as vertentes em que podem ser aplicadas), e constatei que existem várias redes muito utilizadas e que reúnem indicadores já claramente testados e validados, como por exemplo: redes *LeNet, AlexNet, VGGNet, GoogleNet* ou *ResNet*.

Dessa forma, decidi aplicar uma rede criada e formulada apenas “na minha perspetiva” de resolução do problema. E ainda decidi aplicar duas das redes indicadas no parágrafo anterior: *AlexNet* e *VGGNet*. Seguidamente, é mencionada uma pequena contextualização destas duas redes.

A primeira rede *Deep Leaning* criada já remonta ao ano de 1998, quando Yann Lecun, formulou a rede LeNet-5. Esta rede serviu como apoio a todas as redes que foram posteriormente criadas. A explicação do conceito de rede convolucional, geralmente é apoiada com a contextualização da sua abordagem. Isto é, *Lecun* formulou uma rede constituída por um conjunto intercalado de redes convolucionais e de camadas de *pooling,* ao todo 7 camadas. Sendo que, as primeiras camadas convolucionais são representadas por baixos valores de filtros e altos valores de *kernel.* À medida que se avança na rede existe uma inversão desta definição, isto é, existe um aumento do nº de filtros e uma redução do tamanho de *kernel* a aplicar. O principal objetivo desta abordagem, é utilizar as 1ª camadas convoluccionais para extrair *features* menos complexos, e nas camadas posteriormente, *features* mais complexas.

A rede *AlexNet,* foi a primeira rede aplicada após a rede criada por *LeCun,* cerca de 14 anos depois (2012). A sua formulação foi muito importante, pois foi graças a Alex Krizhevsky, Geoffrey Hinton, e Ilya Sutskever, que foi possível “reerguer” o conceito de *Deep Learning*. A sua arquitetura apresenta muitas semelhanças, quando comparada com a *LeNet,* como a utilização intercalada de redes convolucionais e de *pooling*,ouo aumento do nº de filtros e a redução do tamanho do *kernel* à medida que a profundidade da rede aumenta. Contudo, foram introduzidas algumas diferenças que foram importantes, na otimização dos modelos, como: a utilização da camada *Dropout*, que diminui a probabilidade dos modelos sofrerem de *overfitting,* a inclusão da função de ativação retificação linear (reLu), sendo esta hoje em dia um *standard,* ou até mesmo a introdução do conceito de *data augmentation,* que é um conceito muito importante que ajuda a melhorar o processo de aprendizagem dos modelos, recorrendo à introdução de novas imagens, que são iguais às já existentes, mas contendo apenas algumas diferenças, como a aplicação de rotações, aumento do brilho, etc.

Por último, a rede *VGGNet* foi uma rede introduzida dois anos após a criação da rede *AlexNet* (2014), por Simonyan e Zisserman. Ao contrário da rede *AlexNet,* que manteve a mesma estrutura da rede *LeNet,* esta rede trouxe uma arquitetura distinta das redes anteriores. Simonyan e Zisserman introduziram o conceito de *Stacked CNN.* Isto é, ao contrário das redes anteriores, não existe uma rede composta essencialmente por camadas convoluccionais sobrepostas por outras camadas (a *AlexNet* considera apenas numa 1ª parte). Para além disso, o tamanho do *kernel,* teve uma redução acentuada, sendo esse valor de 3\*3. Estas redes, são muito utilizadas quando se pretende recorrer à extração de *features,* devido ao seu elevado nº de filtros. Contudo, é necessário ter em consideração que o nº de parâmetros da rede é muito elevado, aumentando assim o poder computacional e o tempo necessário para a sua aplicação.

***Shallow vs Deep Network***:

Inicialmente, foi efetuada uma análise mais incisiva ao problema, e ao tipo de modelo que melhor se adequa à sua resolução. Ou seja, foi considerada a aplicação de dois modelos CNN (abordagem do tipo *AlexNet*) ao problema.

O 1º modelo descreve uma rede *deep* CNN, isto é, uma rede profunda agregando um enorme conjunto de parâmetros de treino. Já o 2º modelo, contrariamente define uma *shallow* CNN, descrevendo um conjunto de dados de treino bem menor.

Esta análise revela-se importante na medida em que permite verificar, se o modelo requer uma rede mais complexa, ou não, para a correta aprendizagem e resolução do problema.

O 1º modelo criado é composto por uma camada convolucional inicial contendo 32 filtros e um *kernel (3,3).* Por uma 2ª camada convolucional composta por 64 filtros, e seguidamente por uma *stacked* CNN agregando 3 camadas convolucionais, cada uma delas composta por 128 filtros. Depois, ainda é representada por 2 *Dense Layers*, cada uma delas composta por 192 neurónios. O número total de parâmetros de treino é de 21 milhões.

As Figuras 1 e 2 ilustram respetivamente a variação do custo e da *accuracy*, ao longo das *epochs*.

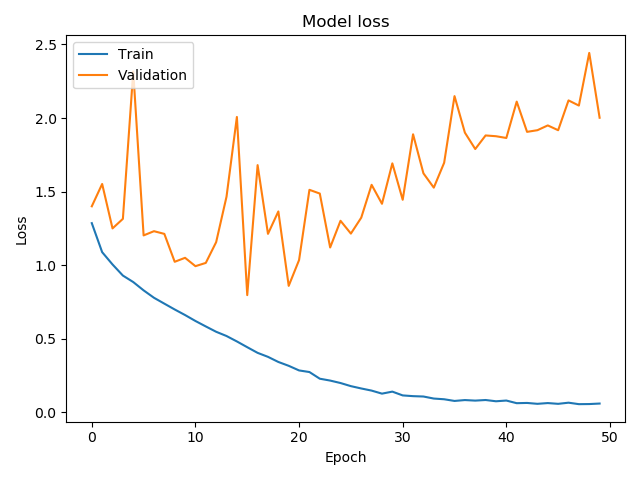


Figura - Variação do custo

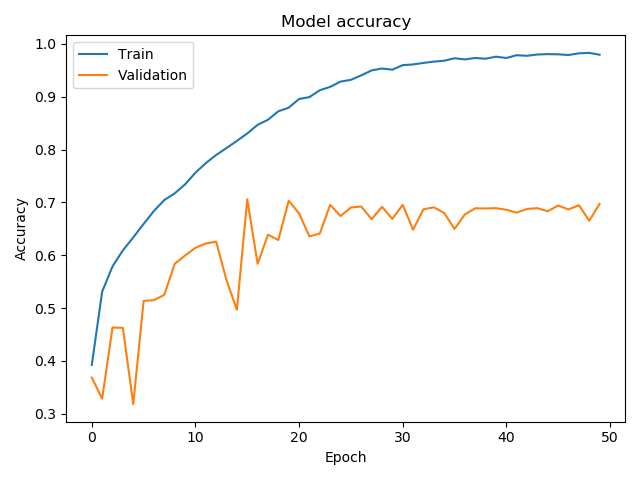


Figura - Variação da accuracy

Observando, ambas as Figuras é possível concluir que o modelo sofre de *overfitting.* Ou seja, o modelo revela-se demasiado complexo, face ao problema a resolver. Uma das abordagens, a utilizar para evitar este efeito de *overfitting*, passaria pela redução do nº de filtros e/ou de neurónios da rede CNN, ou criar uma rede CNN que não seja tão profunda.

Já o 2º modelo criado revela-se bem mais simples, contendo apenas 232 mil parâmetros de treino. Para tal, a rede é composta por: apenas duas camadas convolucionais composta por um *kernel=(3,3)* e um nº de filtros de 8 e 16 respetivamente. É ainda composto por uma *Dense Layer,* que reúne 16 neurónios.

As Figuras 3 e 4 ilustram respetivamente a variação de custo e de *accuracy*, ao longo de 50 *epochs*.

Analisando ambas as Figuras, é possível concluir que a variação quer do custo quer da *accuracy,* se revelam bem mais ajustadas, quando é considerando um modelo mais simples. Ou seja, para a resolução deste problema, deve-se considerar modelos que contenham um nº de parâmetros moderado, visto que comparando os resultados obtidos através das Figuras 3 e 4, com os das Figuras 1 e 2, é possível concluir que ao fim de 50 *epochs* a utilização de um nº mais baixo de parâmetros de treino garante um modelo bem menos tendencioso, e sem problemas de *overfitting*.

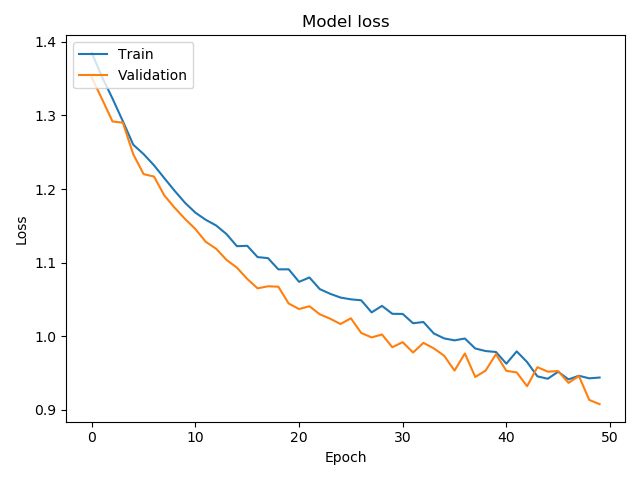


Figura - Variação do custo

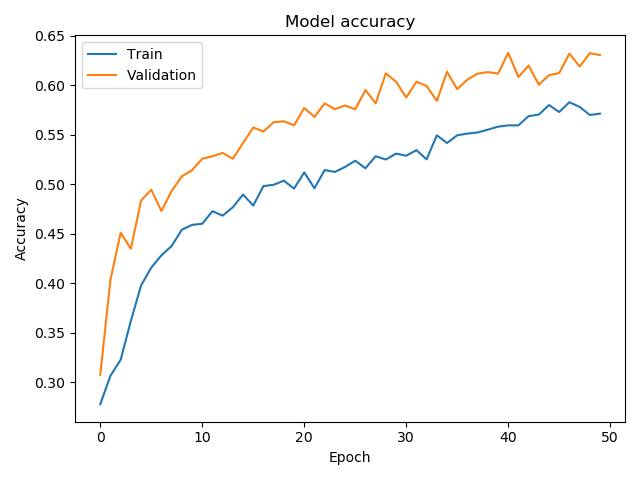


Figura - Variação da accuracy

Contudo, o exemplo descrito seguidamente demonstra que é possível obter resultados interessantes considerando uma rede com um elevado conjunto de parâmetros. Sendo que, para tal é necessário definir corretamente a arquitetura do problema, de modo a que seja possível evitar possíveis problemas de *overfitting*.

Isto é, ao longo de todo o documento são explorados vários modelos que consideram diferentes abordagens, e que permitem a obtenção de bons resultados, ou seja, é possível bons indicadores independentemente do nº de parâmetros considerados, ou se a rede é mais profunda ou não. Para tal, é necessário que o utilizador tenha em consideração, o nº de dados que têm à sua disposição, visto que quando maior o nº de parâmetros de treino maior é a necessidade de possuir muitos dados. E deve ainda tentar identificar se existe ou não necessidade de possuir uma rede muito pesada ou não, ou seja, o utilizador deve tentar otimizar sempre o tempo e poder computacional exigidos na execução do modelo, tentando assim identificar o modelo que se revela mais “leve”, e que se ajusta adequadamente ao problema a resolver.

**Aplicação sem otimização da rede *AlexNet****:*

Inicialmente, e de modo a que fosse possível evidenciar o comportamento da rede na resolução deste problema, foi considerada a aplicação da rede sem recurso a qualquer mecanismo de otimização. Os valores referentes aos seus hiperparâmetros, foram identificados recorrendo à sua alteração manual, tentando assim ajustar estes valores, de uma forma manual. De salientar ainda, que foi necessário recorrer à utilização de valores baixos de *epochs* devido à exigência computacional e de tempo, na sua aplicação.

Dessa forma, foram considerados valores por *default,* para os filtros das camadas convolucionais e ainda para o nº de neurónios das *Dense Layers*. Nomeadamente, 96, 256 e 384 para as 4 camadas convolucionais aplicadas (384 para as 2 *Stacked*). Já o nº de neurónios definido para as três *Dense Layers* foi de 4096, 2048 e 1024 respetivamente. Já os valores de *kernel* e de *pool-size* seguiram os padrões, considerando apenas pequenas alterações. O nº de *epochs* considerado foi baixo, tal como já referido, sendo este de 50 *epochs*.

Relativamente aos resultados obtidos, foi possível comprovar que ao fim de 50 *epochs,* o modelo ainda estava a sofrer de *underfitting*, visto que a *accuracy* de treino ainda rondava os 75%, estando assim ainda longe dos expectáveis 100%. Dessa forma, importa salientar que o modelo necessitaria de ser submetido a mais treino, de modo a que o mesmo conseguisse aprender corretamente os dados, para que depois o *predict* fosse condizente com a aprendizagem aferida pelo modelo.

Na Figura 5, é possível visualizar os resultados obtidos ao fim das 50 *epochs*.



Figura - Resultado obtido considerando 50 Epochs

É possível visualizar que ao fim das 50 *epochs,* o *loss* relativo ao treino e à fase de validação são muito próximos, revelando-se assim um bom indicador. Isto é, significa que o modelo não está a sofrer de *overfitting*.

Para além disso foi ainda considerada a aplicação de um modelo considerando 100 *Epochs,* sendo que neste exemplo foi efetuada uma análise ao comportamento da variação do *loss* e da *accuracy,* entre o treino e a validação. Da aplicação do modelo resultaram os seguintes gráficos, no final das 100 *epochs.*

Da análise às Figuras 6 e 7 é possível comprovar que até às 60 *epochs,* o modelo têm um comportamento adequado, isto é não existe *overfitting.* A partir das 60 *epochs*, a validação começa a estagnar deixando de existir evolução, não fazendo assim sentido continuar a execução do modelo. Relativamente, aos resultados obtidos, a *accuracy* de treino no fim das 100 *epochs* foi de 79%, enquanto que a de validação foi de 76,9%.

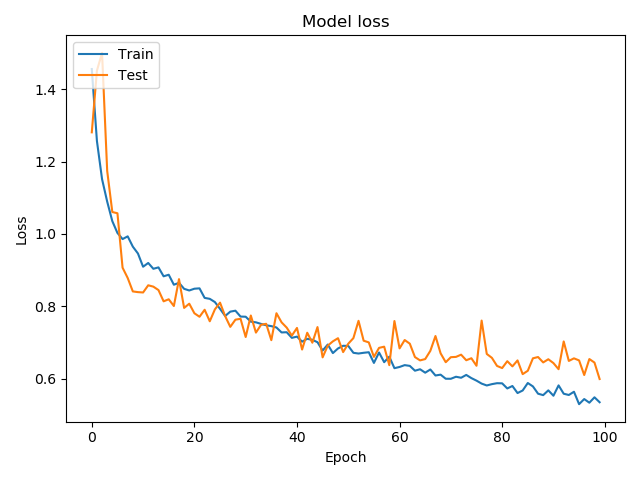


Figura – Variação da loss entre Treino e Validação

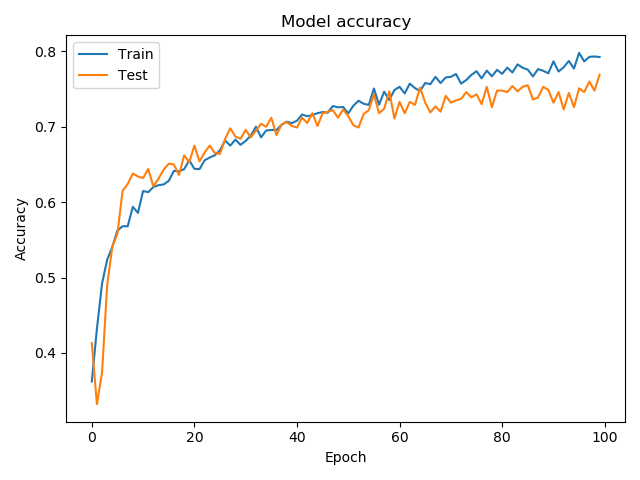


Figura - Variação da accuracy entre Treino e Validação

Os resultados obtidos foram bem interessantes, recorrendo à implementação de uma rede *AlexNet* profunda e com um elevado conjunto de parâmetros de treino. Contudo, no exemplo demonstrado seguidamente, é considerada a aplicação de um modelo baseado também na arquitetura *AlexNet,* mas que reúne um conjunto de dados de treino bem menores permitindo assim reduzir o tempo e custo computacional exigidos na sua aplicação.

**Aplicação faseada do modelo (*save and load model – application of Augmentation*)**:

Outra das abordagens muito utilizadas pelos *data scientists*, assenta na aplicação faseada de modelos. Ou seja, em diversas situações é necessário proceder à salvaguarda do modelo obtido. Por exemplo, para aplicar em contextos preditivos, sem necessidade de voltar a treinar e avaliar o modelo. Mas, é um processo também vital quando se pretende reduzir o impacto computacional exigido no treino do modelo (falta de requisitos do sistema), ou para definir novas configurações de treino do modelo, depois de aplicado um treino ao modelo.

Neste exemplo, recorreu-se a esta abordagem de modo a garantir que o custo e a *accuracy* do modelo estagnassem (não considerando o treino do modelo com *augmentation)*, o modelo era guardado, sendo posteriormente carregado, e depois o mesmo era treinado recorrendo ao conceito de *augmentation*.

O principal objetivo, passa essencialmente pela tentativa de obter um modelo que consiga resistir aos efeitos de *overffiting,* que são obtidos ao fim de um nº de *epochs,* quando não é considerada a aplicação de *augmentation*.

Recorrendo à visualização das Figuras 6 e 7, conseguimos identificar que o modelo ao fim de 150 *epochs* tende a estagnar, contudo a fase de treino ainda se encontra distante dos 100% de *accuracy* esperados.

Dessa forma foi criado um modelo mais simplista, do que o modelo descrito anteriormente, e que permite a obtenção de resultados tão ou melhores, do que os apresentados. A Figura 8 descreve a arquitetura considerada.



Figura – Parâmetros da rede – 32,64 e 64 são respetivamente o nº de filtros das camadas convolucionais aplicados, já 32 e 16 representam o nº de neurónios relativos às Dense Layers

Sendo assim, uma das abordagens consideradas foi a aplicação do conceito de *data augmentation*, no modelo obtido ao fim das 150 *epochs*. Com o objetivo, de evitar a estagnação do modelo, tentando que ainda surja um aumento da aprendizagem e generalização do modelo, isto é, uma subida da *accuracy* e descida do custo, em ambas as fases (treino e validação). As Figuras 9 e 10 descrevem respetivamente a variação da *accuracy* e do custo ao fim de 150 *epochs*.

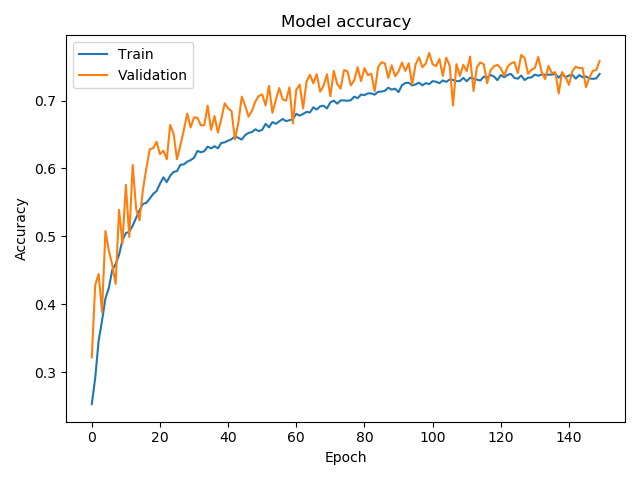


Figura – Variação da accuracy – accuracy final de 76% (validação)

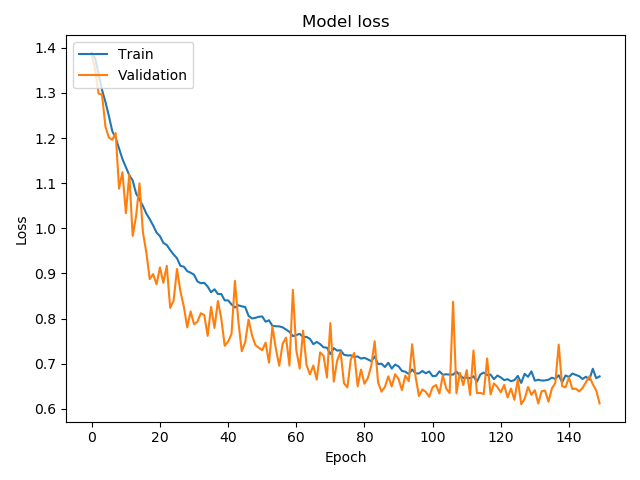


Figura – Variação do custo – custo final de 0.61 (validação)

Posteriormente, foi aplicado então o conceito de *Data Augmentation* ao modelo salvaguardado, e mencionado atrás. Foram apenas consideradas ligeiras alterações nas imagens “originais” do *dataset.* Isto é, são inúmeras as possibilidades que o utilizador possuiu para conseguir aumentar o nº de imagens e ainda a variação imposta nos dados. Ou seja, o utilizador pode aumentar os dados considerando várias técnicas para esse efeito tais como: *Flip,* Rotação, *Crop,* adição de ruído, alteração do brilho, entre outros. Cabendo, ao utilizador perceber quais as técnicas que melhor se adequam ao problema em questão. Por exemplo, considerando um problema de Reconhecimento de Dígitos, é notório que considerar um *Flip* na vertical, pode introduzir erros, e leva a que a aprendizagem seja incorreta. Pois, ao efetuar um *flip* na vertical do nº 6 obtenho o nº 9, e o modelo aprende a reconhecê-lo como um 6, o que não representa a realidade.

Mas, quando aplicado em conformidade, permite ao utilizador a obtenção de modelos mais precisos e menos tendenciosos, essencialmente, porque ao introduzir uma panóplia de dados maior permite ao modelo melhor o seu processo de aprendizagem (aprende um maior nº de situações possíveis), e por sua vez garante uma generalização mais precisa.

No que toca, ao problema em estudo, foram consideradas as seguintes técnicas de *Data Augmentation*:

* Horizontal *Flip;*
* Deslocamento dos pixéis na Largura em 10%;
* Deslocamento dos pixéis na Altura em 10%;
* Zoom da imagem em 10%;
* *Shear Range,* em 10%;

São muito mais as opções, que poderiam ser consideradas neste processo de transformação, contudo, e considerando este problema, as mencionadas são aquelas que melhor se ajustam ao problema.

Seguidamente, as Figuras11 e 12 descrevem a variação do custo e da *accuracy*, do modelo salvaguardado. Dessa forma, as *epochs* descritas em ambas as Figuras podem ser vistas como uma adição face às *epochs* já realizadas, e visíveis nas Figuras 9 e 10.

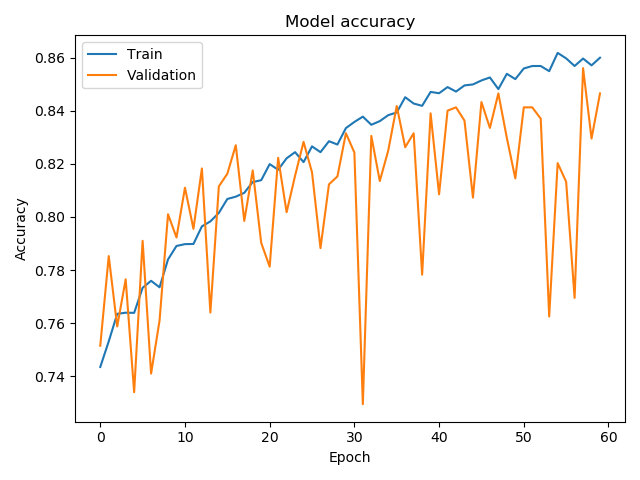


Figura – Variação da accuracy

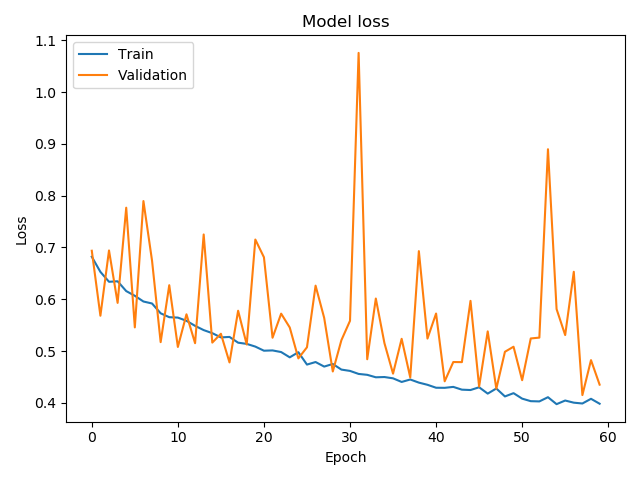


Figura – Variação do custo

A aplicação de *Data Augmentation* trouxe uma melhoria, quer do custo, quer da *accuracy.* Através da análise das Figuras 9 e 10, foi possível identificar que ao fim de 120 *epochs*, a aprendizagem do modelo começava a estagnar. Sendo assim necessário promover alguma alteração, de modo a que fosse possível evitar essa estagnação. A solução passou pela introdução de um maior conjunto de dados no problema, permitindo assim otimizar o processo de aprendizagem, evitando assim que o mesmo estagnasse. Observando, as Figuras 11 e 12 é possível confirmar a melhoria contínua, da *accuracy* e do custo.

Sendo assim, é possível concluir que a utilização de *Data Augmentation,* revela-se vantajosa em situações, em que à primeira vista, já não é possível otimizar mais o modelo. Para além disso, esta metodologia permite a redução dos tempos exigidos na execução dos modelos, visto que inicialmente procede-se ao treino do modelo, não considerando *Data Augmentation,* e só quando o mesmo tende a estagnar é que se aplica *Data Augmentation*. Permitindo assim, que haja um controlo gradual do processo de treino e de generalização do modelo.

Considerando, o exemplo descrito em cima, é possível concluir que recorrendo ao conceito de *Data Augmentation* foi possível otimizar ainda mais o modelo, em cerca de 9%. Sendo que, o treino inicial, ao fim das 150 *epochs* demonstrou que o modelo estagnou nos 76%, e com a aplicação de *Data Augmentation* foi possível obter um modelo com uma *accuracy* de 85%. Este valor, provavelmente ainda aumentaria, visto que o modelo ainda tem margem de progressão para aprender melhor os dados.

**Aplicação do PSO, como otimizador da rede**:

Seguidamente, decidi utilizar o algoritmo *Particle Swarm Optimization,* de modo a que fosse possível identificar os melhores valores para alguns dos hiperparâmetros do modelo. Para tal, decidi recorrer à otimização do nº de filtros das várias camadas que compõem a rede CNN, e ainda a otimização do nº de neurónios das *Full-Connected-Layers* utilizadas. Para além destes hiperparâmetros, é habitual a otimização de mais parâmetros, tais como: *batch size*, o valor do *learning rate* relativo ao otimizador utilizado, ou ainda a função de ativação utilizada nas várias camadas da rede. As referências [2] e [3] exemplificam a abordagem utilizada, e os híper-parâmetros da rede, que normalmente são considerados no processo de otimização. **De realçar, que da pesquisa realizada, os algoritmos de otimização são utilizados para identificar valores mais adequados para uma determinada rede, e não para determinar qual a rede mais apropriada para a resolução de um problema**.

Sendo assim, foi aplicado o PSO na tentativa de identificar os valores mais adequados para os parâmetros referidos no parágrafo anterior. Na sua aplicação foram apenas consideradas 15 partículas, 2 iterações e 5 *epochs*, visto que o poder computacional e tempo exigido na sua execução são elevados. A Figura 13 demonstra os valores inerentes ao nº de filtros de cada camada convolucional (3 primeiros valores) e restantes dois valores representam o nº de neurónios para as duas *Dense Layers* utilizadas.



Figura - Resultado obtido considerando aplicando PSO - 15 Partículas, 2 Iterações e 5 Epochs

De modo a comprovar os resultados obtidos com a aplicação do PSO, foi aplicado o modelo considerando os valores indicados pelo PSO. De realçar que não é expectável a introdução de melhorias muito significativas, ou até nenhumas, visto que o algoritmo foi aplicado considerando um baixo nº de partículas e de iterações, reduzindo assim a viabilidade dos resultados obtidos.

Já recorrendo à aplicação de um nº mais elevado de iterações (5 iterações) e um maior nº de *epochs* (10 *epochs),* foi obtido um custo final de 5.35, considerando a topologia *Global Best*. A Figura 14 demonstra os resultados obtidos.



Figura – Custo e posição obtidas, após aplicação do PSO

Foi ainda, definida uma rede CNN constituída pelos valores identificados pelo PSO, na tentativa de aferir os resultados obtidos. Os resultados obtidos foram muito prometedores ao fim de 30 *epochs*. Através das Figuras 15 e 16 é possível perceber que o modelo apresenta uma variação adequada quer do custo, quer da *accuracy*,em ambas as fases de treino e de validação. Até ao momento, o modelo não revela sinais de *overfitting* ou de *underfitting,* demonstrando uma queda gradual de ambas as variáveis, e a diferença de valores entre ambas as fases é idêntica, revelando assim uma “boa” capacidade de aprendizagem e de generalização.

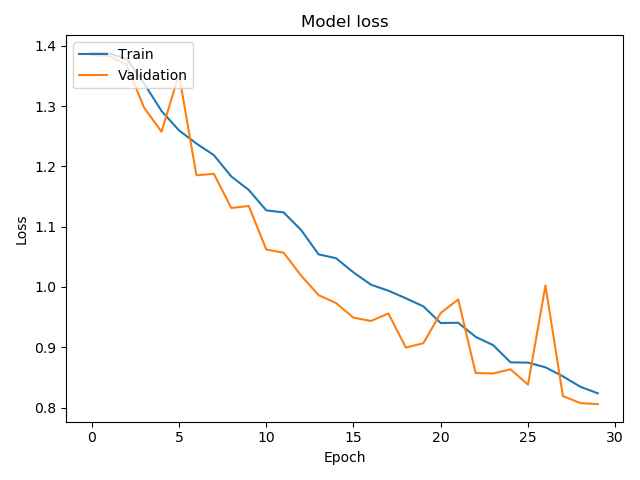


Figura - Variação do Custo

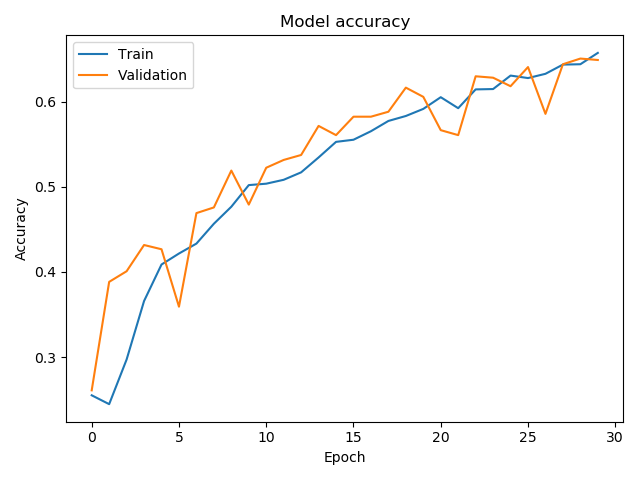


Figura - Variação da accuracy

**Aplicação do *Genetic Algorithm****:*

Para além da aplicação do PSO, foi ainda aplicado outro algoritmo evolucionário, neste caso o *Genetic Algorithm*. De realçar que foi considerada a aplicação da biblioteca DEAP. Da pesquisa realizada, o autor chegou à conclusão que esta biblioteca é aquela que proporciona maior flexibilidade ao utilizador, apesar de requerer uma maior complexidade a nível de código.

O processo de aplicação do GA envolve uma série de passos iniciais que são necessários, e que necessitam de ser devidamente ponderados. É necessário determinar como é que a solução deve ser representada. Neste caso foi considerada a sua representação recorrendo ao formato de um *array* binário. Onde o seu tamanho é determinado considerando uma divisão do *array*, pelos vários parâmetros a otimizar. Isto é, caso pretenda otimizar dois parâmetros, sendo que o primeiro pode ser representado por 64 valores e o 2º por 16, então o array tem tamanho = 10. Os 6 1º *bits* são referentes ao 1º parâmetro a otimizar e os 4 restantes *bits* ao 2º parâmetro a ser otimizado. A Figura 17 descreve a abordagem utilizada.



Figura – Representação da Solução

No problema a resolver foi considerada a utilização de um *bit* de tamanho x. Sendo que o nº de filtros relativos às redes convolucionais, são representados respetivamente pelos 6 primeiros dígitos, 7 dígitos seguintes, e 8 dígitos seguintes. Já o nº de neurónios, presente em cada *Dense Layer* é representado respetivamente pelos 8 e 6 dígitos. No total a representação da solução incorpora um total de 35 dígitos (6+7+8+8+6).

Para além disso, foi necessário prestar ainda atenção a dois pontos fulcrais dos Algoritmos Genéticos, isto é, o operador de mutação e de *crossover*. Relativamente, ao operador de mutação, foi considerado a mutação aleatória de índices de uma solução, com uma probabilidade de 0.2. Já na operação de *crossover,* foi considerada a opção de *crossover* ordenado, com uma probabilidade de 0.4.

Posteriormente, foi aplicado o algoritmo, sendo que para tal foram considerados valores baixos da população e do nº de gerações, devido ao poder computacional e tempo exigidos, na sua aplicação. Dessa forma foi considerada uma população igual a 10 e um nº de gerações igual a 5. A aplicação do GA indicou que o nº ideal de filtros para as camadas convolucionais da rede são respetivamente: 57, 114 e 195. Já o nº ideal de neurónios, é respetivamente: 195 e 33.

De modo a confirmar, o comportamento do modelo quando exposta a estes valores, seguidamente procedeu-se à execução do modelo, considerando os valores referidos atrás.

As Figuras 18 e 19 demonstram respetivamente a variação do custo entre o treino e a validação, e ainda a variação da *accuracy* entre ambas as fases.

Da análise a ambas as Figuras é possível concluir que os resultados obtidos foram interessantes, visto que quer o custo quer a *accuracy* em ambas as fases, encontram-se próximas entre si. Ao fim das 30 *epochs* é possível verificar que o modelo ainda está longe de apresentar a aprendizagem concluída, estando a mesma ainda próxima dos 65%. É expectável que a *accuracy*,à medida que as *epochs* vão avançando, que acompanhe a *accuracy* de treino. Importa ainda referir que o GA foi aplicado considerando nº baixos de população, gerações e de *epochs,* caso estes valores fossem mais elevados e condizentes com o problema, era expectável que os resultados obtidos, fossem ainda melhores.

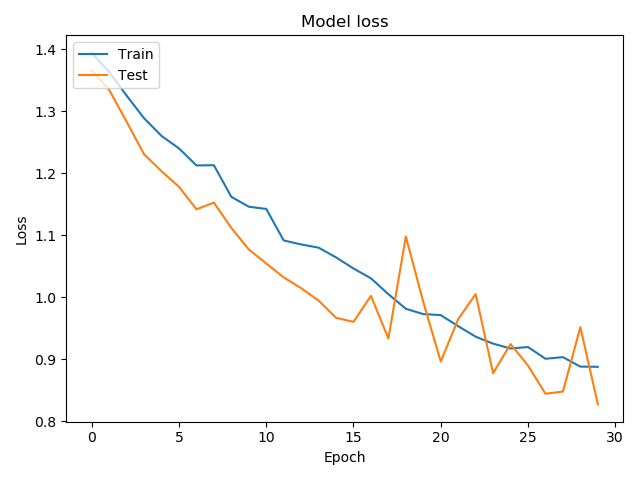


Figura – Variação do custo, na Fase de Treino e de Validação

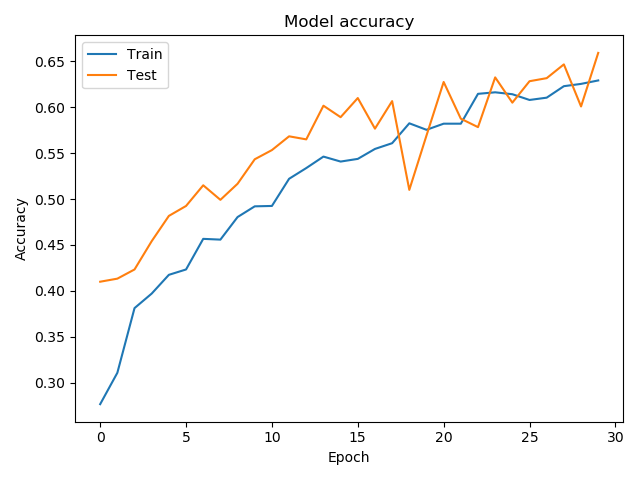


Figura – Variação da accuracy, na Fase de Treino e de Validação

**Aplicação do tipo de rede *VGG-Net****:*

Para além da aplicação da aplicação de redes baseadas em *AlexNet,* foi ainda considerada a aplicação de redes baseadas em *VGGNet*.

O tipo de análise descrita posteriormente, apresenta um carácter mais simples e mais objetivo, face à abordagem contextualizada anteriormente. As redes *VGG* requerem maior poder computacional e temporal para a sua execução, e como o documento já é longo e a análise profunda, seguidamente irão ser apenas descritos e explicados os resultados obtidos, através da aplicação de uma rede *VGG* mais profunda, e os resultados obtidos considerando uma rede mais “leve”.

Tal como já fora mencionado atrás, as redes *VGGNet* são reconhecidas essencialmente pela sua arquitetura “muito bem” organizada, e ainda pelo elevado nº de parâmetros de treino/não treino que frequentemente possuem.

O 1º modelo aplicado apresenta um teor mais simplista, isto é, contêm um menor nº de camadas convolucionais. A rede considerada é composta por 3 *stacked convolution layers*, ou seja, a 1ª *stacked layer* é composta por duas camadas convolucionais contendo 32 filtros, já a 2ª *stacked layer* é também ela composta por duas camadas convolucionais, sendo o seu nº de filtros igual a 64. Já a 3ª e última *stacked layer* é composta por três camadas convolucionais, cada uma delas contendo 128 filtros. No fim, recorre-se à utilização de uma única *Dense Layer* composta por 32 neurónios. Da aplicação desta rede resultam 566 mil parâmetros.

De realçar que, recorreu-se à utilização de *Data Augmentation*,tal como sugerido e utilizado na aplicação “oficial” da rede *VGGNet*.

As Figuras 20 e 21 ilustram respetivamente, a variação do custo e da *accuracy*, ao fim de 100 *epochs*.

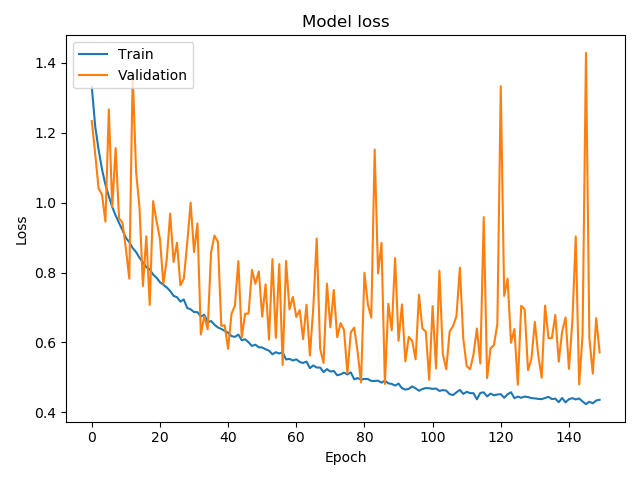


Figura - Variação de custo

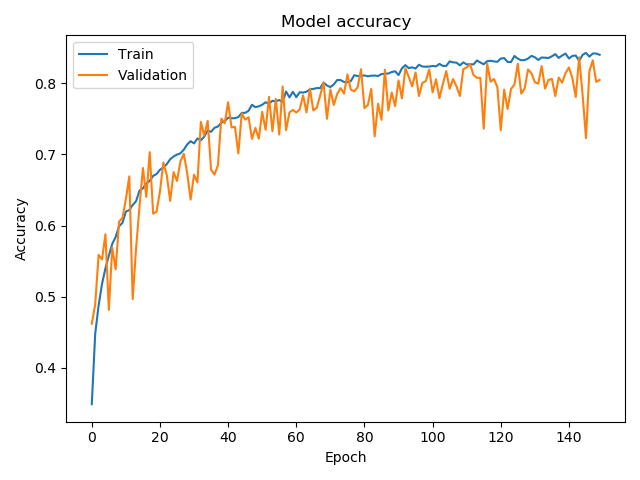


Figura - Variação da accuracy

Analisando as Figuras 20 e 21 podemos concluir que o modelo apresentou resultados bem interessantes. O modelo ao longo das 150 *epochs* não revelou problemas de *overfitting*, visto que o custo e a *accuracy* de validação acompanharam, ao longo das várias *epochs* o treino. Contudo, é possível concluir que o modelo ao fim das 120 *epochs* tende a estagnar a sua aprendizagem, visto que o custo de treino tende a estagnar, estando diretamente relacionado com a complexidade da arquitetura considerada e/ou nº de amostras utilizadas na fase de treino. Ou seja, apesar do modelo ainda possuir margem de melhoria, a melhoria não será muito notória, visto que a *accuracy* de validação, já se encontra na fase de estagnação. Ainda assim, este modelo permitiu a obtenção de uma *accuracy* de teste de 81%.

Tal como fora referido anteriormente, foi considerada a aplicação de um modelo mais complexo, com o objetivo de analisar os resultados obtidos quando considerada uma rede *VGG*, mais complexa neste problema.

Neste 2º modelo foram aplicadas cinco *stacked convolution layers*, sendo que o nº de filtros considerado em cada *stack* também foi maior comparando com o modelo anterior. A 1ª *stack* é composta por 64 filtros, a 2ª por 128, a 3ª por 256 e a 4ª e 5ª por 512 filtros. Já a *Dense Layer* é constituída por 512 neurónios.

As Figuras 22 e 23 ilustram respetivamente a variação do custo e da *accuracy*, ao longo das 150 *epochs* consideradas.

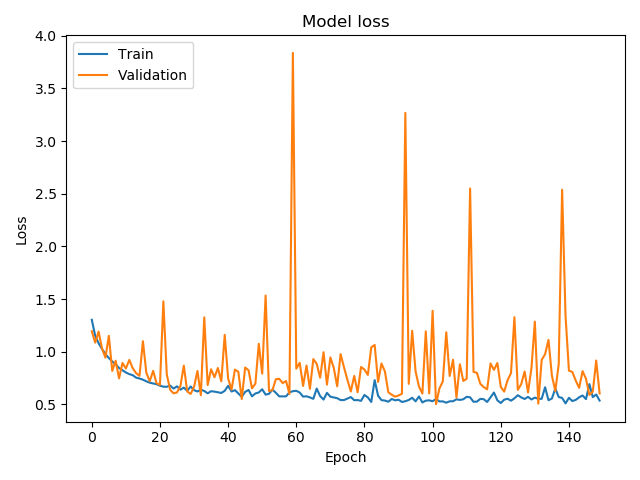


Figura - Variação do custo

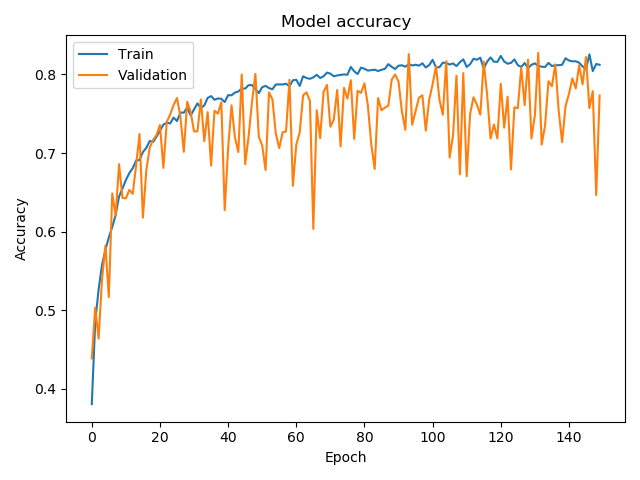


Figura - Variação da accuracy

Da análise a ambas as figuras é possível concluir que a aplicação de uma rede *VGG* mais complexa, comparativamente com a aplicada anteriormente, revela resultados semelhantes aos obtidos atrás. Contudo, existem algumas diferenças, nomeadamente a estagnação mais precoce deste modelo. Isto é, o modelo ao fim das 80 *epochs*, já se encontrava estagnado, sendo muito difícil evidenciar a descida do custo e a subida da *accuracy.* Contudo, o modelo ao fim de 150 *epochs* não apresentou sinais de *overfitting*. A *accuracy* final de teste revelou-se mais baixo que a obtida na aplicação do modelo anterior, 77%.

A aplicação do 1º modelo revelou-se mais eficaz, essencialmente porque o modelo apresentou uma *accuracy* de teste superior na ordem dos 81%, para além de que estagnou mais tardiamente do que o 2º modelo. Outro fator importante, é a sua menor complexidade que permite a sua execução num menor espaço de tempo, exigindo assim menor custo computacional.

**De modo, a tentar otimizar ambos os modelos, foi considerada a aplicação de ambos os modelos, tendo em conta um maior nº de dados para treino. O objetivo passa pela tentativa de melhorar os resultados obtidos e a aprendizagem dos modelos. Para além disso, de modo a verificar qual a rede que melhor se ajusta ao problema (rede mais simples, ou rede mais complexa).**

Até então, a fase de treino dos modelos teve em conta a utilização de 3000 imagens. Contudo, ainda existem 2000 imagens restantes para a fase de treino. Essas imagens, serão utilizadas seguidamente, com o objetivo de perceber qual a influência que o aumento do nº de imagens têm nos modelos, mais concretamente se ajudam ambos os modelos contextualizados atrás, na obtenção de melhores resultados.

Considerando o 1º modelo (modelo mais simples) é expectável que o modelo apresente uma aprendizagem e generalização melhor ou semelhante à apresentada em cima. Ou seja, considerando um maior nº de *epochs*, espera-se que o modelo consiga apresentar uma aprendizagem e generalização adequadas, permitindo assim a obtenção de bons resultados. Ao considerar um maior nº de exemplos de treino, aplicados ao mesmo modelo, ajuda a evitar a estagnação que começou a ser visível ao fim das 120 *epochs*, visíveis através da Figura 20.

As Figuras 24 e 25 descrevem os resultados obtidos.



Figura - Variação do custo

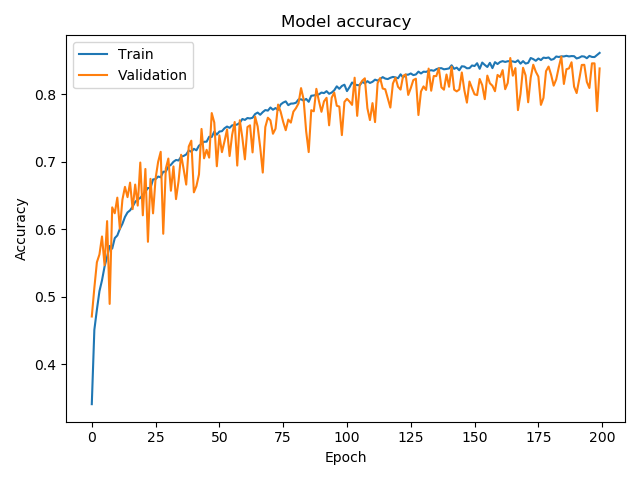


Figura - Variação da accuracy

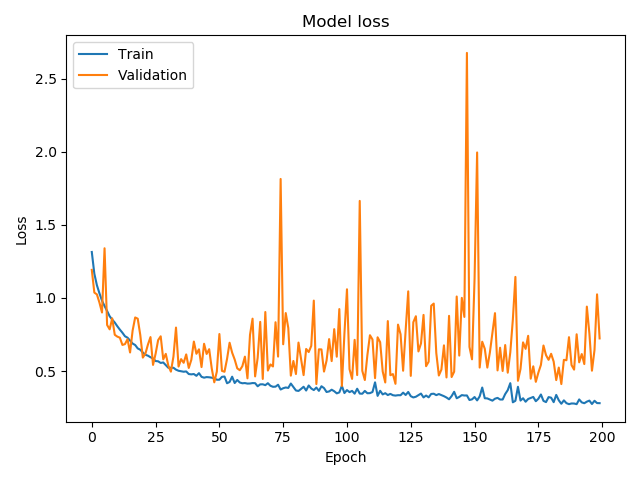
Relativamente aos resultados obtidos, é possível constatar que os mesmos estão em concordância com os resultados já esperados, isto é, revela-se semelhantes aos resultados obtidos, sem a aplicação de *Data Augmentation*. Ainda assim, o modelo apresentou uma ligeira melhoria na *accuracy* final de teste, na ordem dos 3%, passando a mesma a ser de 84%.

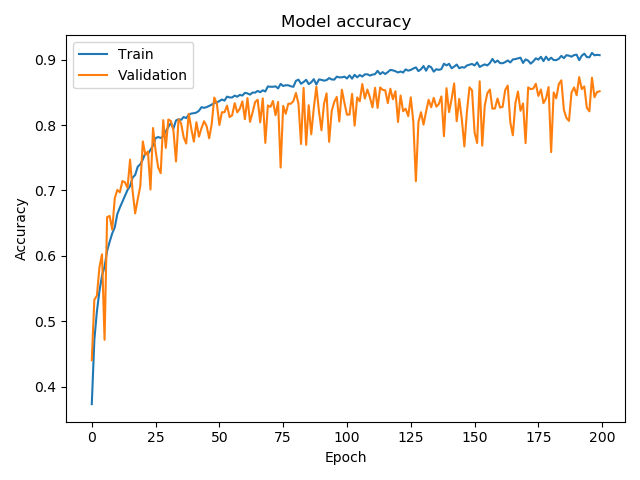
Contudo, e tal como já expectável a introdução de *Data Augmentation,* não proporcionou uma melhoria significativa nos resultados. Visto que, anteriormente o modelo já fora treinado considerando um conjunto de dados muito aceitável, e a introdução de mais dados não influenciaria significativamente os resultados finais obtidos.

Contudo, a obtenção de uma *accuracy* final, na ordem dos 84% revela-se um bom indicador, sendo que este valor ainda poderia ser mais otimizado (necessário estudo intensivo).

Na aplicação do 2º modelo, considerando um maior nº de amostra de treino, é expectável que os resultados obtidos sejam semelhantes aos já obtidos e descritos, através das Figuras 22 e 23. Eventualmente, é também esperado que o modelo apresente algumas melhorias, dado que a adição de um maior nº de amostras, deve permitir ao modelo melhorar a sua fase de aprendizagem, ainda assim não é expectável que surjam melhorias muito significativas, mas sim ligeiras melhorias.

As Figuras 26 e 27 descrevem os resultados obtidos.





Os resultados obtidos foram mais satisfatórios do que o previsto inicialmente. O modelo apresentou uma *accuracy* final de teste, na ordem dos 85%, isto é, a adição de mais dados permitiu a melhoria percentual de 8% face à utilização de um menor conjunto de dados.

Por outro lado, é ainda possível concluir que o modelo ao fim de 100 *epochs* já começou a dar indícios de sofrer de *overfitting*, sendo que o mesmo estagnou ao fim das 100 *epochs*. Este facto pode estar relacionado com: (1) necessidade de mais dados para treino/validação, (2) necessidade de ajuste do *Learning Rate*.

Este modelo revela-se muito complexo, e à medida que o nº de dados aumenta, o poder computacional e o tempo exigido na sua execução aumenta cada vez mais, dificultando assim a análise do modelo. É ainda possível concluir que a aplicação de um modelo *VGGNet* mais simples revela-se mais adequado a este problema, visto que permite a obtenção de melhores resultados, e exige menos poder computacional.

**Aplicação de Rede CNN ao problema *Breast Cancer***:

Considerando o *dataset* utilizado há uns meses, na tentativa de recorrer a técnicas de *feature selection,* na ajuda do estudo e da classificação de um problema de análise da expressão de genes, o mesmo *dataset (Breast Cancer)* foi utilizado e testado recorrendo à aplicação de redes convolucionais.

De salientar, que o *dataset* inicial é composto por 24 amostras (10 casos com doença e 14 sem doença), e ainda é composto por 9670 atributos, neste caso genes. Da aplicação das técnicas de *feature selection, K-Means* e *Binary Particle Swarm Optimization*, o *dataset* foi reduzido para 21 *features.*

Com o *dataset* já devidamente pré-processado e limpo (redução de *features)* foi aplicado um modelo, o SVM, de modo a comprovar se as técnicas de *feature selection,* promoveram vantagens ou não, confirmando com os resultados obtidos. Os resultados foram muito satisfatórios, estando os mesmos compreendidos na gama dos 83% (*accuracy, prediction, recall)*.

Considerando o *dataset* já devidamente reduzido, e de modo a comprovar a utilidade da aplicação das redes convolucionais a este tipo de problemas, foi considerada a sua aplicação a este problema.

Para tal e segundo o estudo efetuado previamente, irão ser consideradas duas abordagens na resolução deste problema: (1) aplicação de uma rede convolucional 1D e (2) aplicação de uma rede 2D.

Seguidamente, segue-se a descrição dos resultados obtidos recorrendo às duas abordagens mencionadas.

**Rede CNN-1D**:

Antes de proceder à criação e execução da rede convolucional, é necessário preparar o *dataset*. Uma vez que é irão ser utilizadas camadas convolucionais 1D, é necessário perceber que tipo de dependências podem existir no *dataset* – as conhecidas *Time Series*.

Relativamente, às amostras sabemos de antemão que não existe qualquer tipo de dependência entre estas, visto que correspondem a pessoas distintas, não existem “leituras” em espaços de tempo distintos para a mesma pessoa. Relativamente, aos genes pode ou não existir uma dependência entre estes. Sendo assim, é importante garantir que o *kernel* de uma rede convolucional, apenas analisa uma amostra de cada vez, e não uma dependência entre estas.

Seguidamente, é necessário recorrer à transformação do *shape* dos *dataset*, isto é, deve ser considerado o *shape* indicado pela documentação *Keras (samples, time steps, features)*.

Finalmente, e após estarem devidamente definidos os dados de treino e de teste do problema, é necessário realizar a operação de *hot-encoding* dos *targets,* associados aos dados de treino e de teste do problema. De modo, a que estejam em conformidade com o resultado retornado pela função *softmax* (função de ativação da última *Full Connected Layer)*.

A rede criada tem necessariamente de ser muito simples, devido ao facto do *kernel* aplicado apresentar uma largura igual ao nº de atributos do *dataset*. Esta abordagem foi considerada, porque neste tipo de problemas faz mais sentido a análise completa de uma amostra, e não a análise faseada de uma amostra. Sendo assim, foi apenas aplicada uma camada CNN, pois o *output* da sua aplicação é reduzido um *shape* igual a (nº de *samples*, 1). Não fazendo dessa forma sentido aplicar camadas convolucionais subsequentes constituídas por um *kernel* de (1,1). Por outro lado, também não faz sentido aplicar camadas de *pooling,* visto que o retorno da aplicação do *kernel* na camada convolucional retorna um único valor para cada amostra.

Da aplicação de uma rede CNN ao *dataset* reduzido foi possível comprovar os resultados já esperados, isto é, resultados próximos dos resultados obtidos da aplicação do modelo *SVM* após a obtenção do *dataset* reduzido. Os resultados obtidos considerando uma divisão dos dados de: 70%, 15% e 15%, respetivamente para as fases de treino, validação e de teste, permite a obtenção final, de 100% e de 0.28 respetivamente para a *accuracy* e custo de treino.

Depois, foi ainda considerada a aplicação do modelo (ajustado), não considerando a redução do *dataset,* ou seja, o *dataset* considerando corresponde ao *dataset* original do problema, constituído por 24 amostras e 9470 *features*.

Dada a maior complexidade do problema, é necessário aumentar a complexidade do modelo, de modo a que exista uma maior probabilidade de o modelo aprender corretamente os dados. Contudo, não é possível aumentar a complexidade do problema, recorrendo à adição de mais camadas convolucionais, ou à adição de camadas de *pooling,* dado ao facto já descrito atrás, impedindo assim o aumento da complexidade do problema dessa forma. A única forma de aumentar a complexidade do problema passa pela adição de mais filtro na camada convoluciona, ou no aumento do nº de neurónios das *Dense Layers.*

Os resultados obtidos com a aplicação do modelo, em torno do *dataset* original, retornaram bons indicadores, nomeadamente uma *accuracy* final de 100% (considerando 5 amostras para teste, 3 validação e 16 para treino).

As Figuras 27 e 28 demonstram respetivamente a *accuracy* e custo obtido e ainda a variação do custo ao longo das *epochs*.

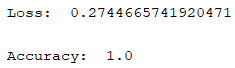


Figura – Resultados obtidos



Figura - Variação do custo ao longo das epochs

**Rede CNN-2D**:

Após a aplicação e descrição dos resultados obtidos, quando aplicada uma rede CNN-1D ao problema, posteriormente e tal como fora referido atrás, foi ainda aplicada uma rede CNN-2D ao problema.

A abordagem aplicada no exemplo anterior, aplica-se de igual forma a este exemplo, sendo que a única diferença, está subjacente ao formato dos dados de entrada, e ainda nos *kernels* das camadas convolucionais, que consideram uma matriz bidimensional, na filtragem dos dados (daí a utilização de uma CNN-2D).

A aplicação de uma rede convolucional 2D, revela algumas vantagens comparativamente à aplicação de uma rede CNN-1D. Visto que, permite ao utilizador a criação de arquiteturas mais complexas e flexíveis, algo que as camadas convolucionais 1D, não permitem. Para além disso, e considerando este tipo de problema em específico, de expressão de genes, torna-se vantajosa a tentativa de identificação de “genes marcadores”.

Neste processo, normalmente recorre-se à ordenação dos genes, considerando os genes que apresentam uma maior dependência entre si, de modo a que o *kernel* consiga identificar zonas que promovam, a identificação de padrões que ajudem o modelo a aprender os dados de uma forma mais correta. Geralmente uma das abordagens consideradas para a ordenação dos genes consiste na disposição destes, de acordo com a ordem posicional dos seus cromossomas[5]. No exemplo criado, foi considerada a ordem posicional dos genes, que fora fornecida inicialmente no *dataset*.

As Figuras 29 e 30 demonstram respetivamente os resultados obtidos e ainda a variação do custo ao longo das *epochs*.

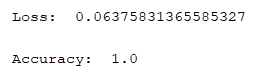


Figura – Custo e Accuracy obtidos

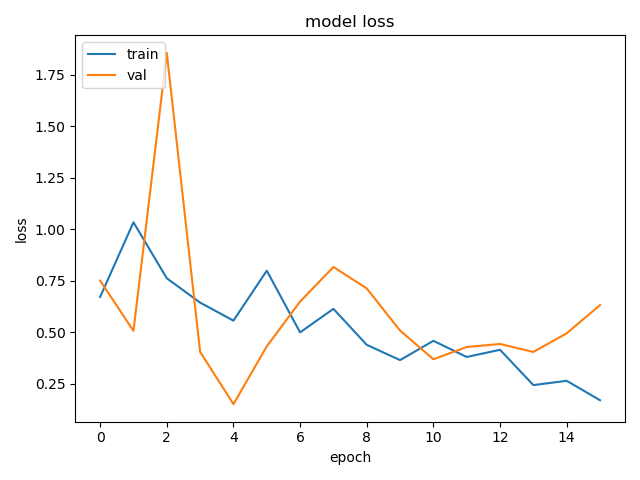


Figura – Variação do custo ao longo das epochs

**Otimização do modelo CNN-2D, com PSO**:

Foi aplicado o algoritmo de otimização PSO, numa tentativa de identificar o nº de filtros e de neurónios mais adequado ao projeto, de modo a que fosse possível obter um modelo mais adequado, isto é, um modelo que não apresenta-se disparidades tão acentuadas entre os custos e *accuracy’s* obtidas na fase de treino e de validação. Para tal, foi considerada uma função de custo, que pretende do mesmo modo minimizar o nº de filtros e de neurónios e maximizar a *accuracy* final. Mas, para além disso pretende-se minimizar as diferenças de custo e de *accuracy,* nas fases de treino e de validação, ao longo das *epochs*, permitindo assim a obtenção de um modelo, que evite *overfitting* e ainda resultados tendenciosos. A Fórmula 1 descreve a função de custo considerada. *MeanDiffLoss* e *MeanDiffAccuracy* representam respetivamente a média da diferença entre o custo e *accuracy* obtidas na fase de treino e de validação, em cada *epoch* considerada.

Porque, mais importante do que garantir um modelo que apresente boas métricas, é importante que os modelos garantam fiabilidade e segurança nos resultados obtidos. Dessa forma, foram introduzidas mais restrições à função de custo.

Na aplicação do PSO foram consideradas 20 partículas e 15 iterações. A Figura 31 demonstra a variação do custo ao longo das iterações. É percetível que as partículas conseguirem minimizar drasticamente o custo inicial, de 4.4 para 3.10, sendo uma redução muito significativa.

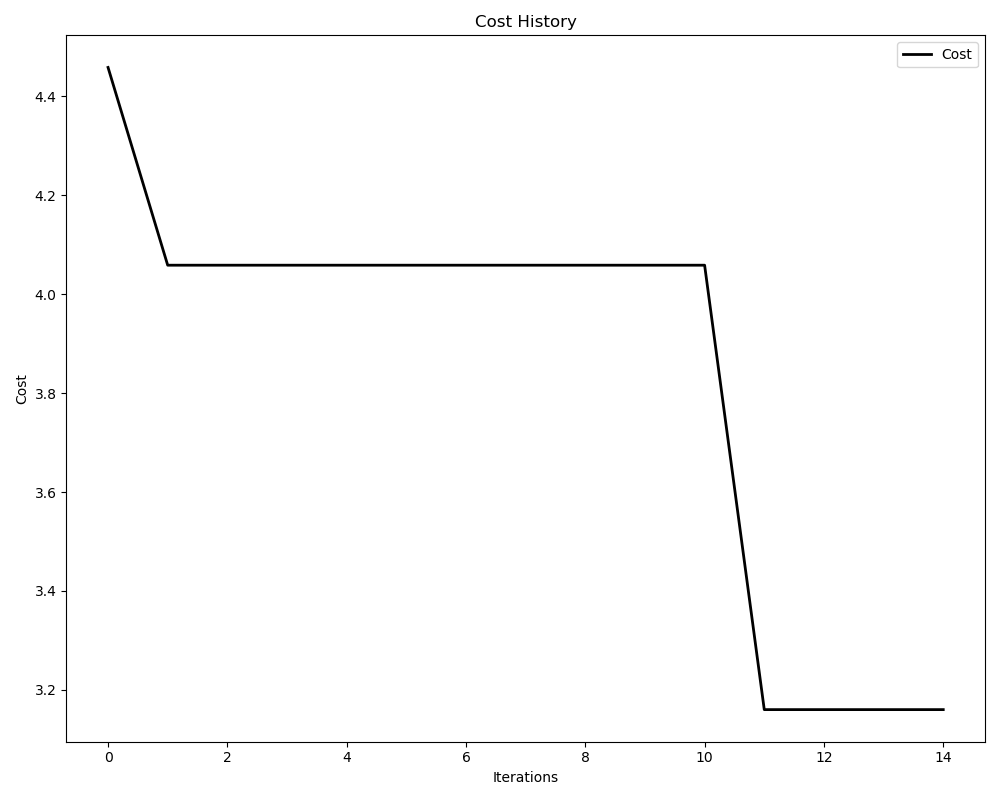


Figura – Variação do custo ao longo das epochs

**Outro aspeto relevante, foi a melhor solução identificado pelo PSO, sendo esta: 21 filtros e 1 neurónio.** Ou seja, a inclusão inicial de uma *Dense Layer,* não se revela neste problema benéfica, sendo que a utilização única de uma rede convolucional permite a obtenção de bons resultados. Mas, de modo a comprovar este facto foi aplicada a rede CNN considerando este nº de filtros e a retirada da *Dense Layer*.

As Figuras 32 e 33 descrevem respetivamente a variação do custo e da *accuracy*, ao longo das *epochs*.

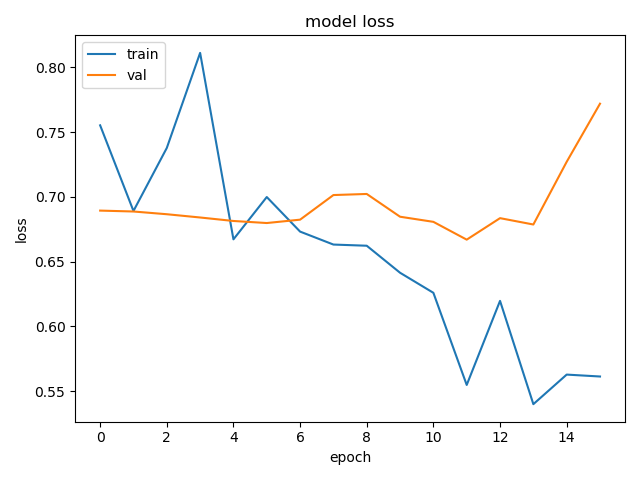


Figura – Variação do custo ao longo das epochs

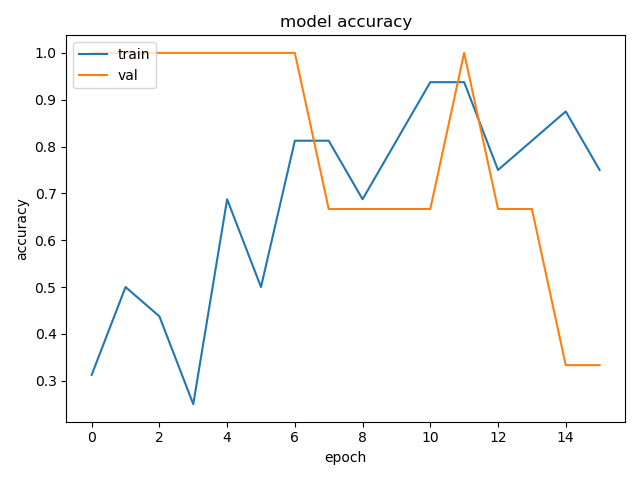


Figura – Variação da accuracy ao longo das epochs

Da sua análise é possível constatar uma melhoria face à variação do custo, ilustrada através da Figura 19. Os eixos encontram-se muito mais compactos (*range* entre 0.55 e 0.8), sendo que a variação máxima é de 0.25. Revelando-se assim um modelo bem mais ajustado, face ao obtido anteriormente. Este facto, deve-se essencialmente à retificação da função de custo, que permite assim a obtenção de um modelo bem menos tendencioso, e revelando maior qualidade e fiabilidade.

Por outro lado, a variação da *accuracy* revelou-se bem mais inconsistente, isto é, da *epoch* 6 até à *epoch* 13, a *accuracy* de treino e de validação revelaram-se bastante próximas. Contudo, nas restantes *epochs* não foi possível verificar essa proximidade. Uma das possíveis razões para esta causa, deve-se ao facto de terem sido consideradas poucas amostras para validação (apenas 4), levando a que possa existir disparidades na *accuracy* de validação.

**Aplicação do PSO e do GA na *Cloud – Cifar-10***:

Os exemplos descritos atrás, referentes à aplicação de ambos os algoritmos evolucionários, *Particle Swarm Optimization* e *Genetic Algorithm*, foram aplicados num contexto pouco “esclarecedor”, isto é, considerando uma otimização muito superficial dos modelos aplicados e descritos anteriormente.

Com o objetivo, de conseguir obter um *feedback* mais preciso das vantagens da utilização de ambos os algoritmos, foi aplicado um exemplo mais rigoroso da sua aplicação. Para tal, foi necessário proceder à criação de uma instância (máquina virtual) na *Cloud*, sendo a mesma implantada recorrendo à *Google Cloud Engine*.

Para tal foi considerada uma máquina *Ubuntu* (Sistema Operativo), composta por oito núcleos e por um total de 30 *gigabytes*, de memória RAM. Este valor parece elevado, mas é devidamente necessário, visto que o poder computacional exigido na utilização quer do PSO, quer do GA requerem uma máquina com uma memória elevado, dado o nº de parâmetros envolvidos no processamento dos modelos (otimização iterativa dos modelos).

De modo, a que fosse possível aumentar a flexibilidade e a interatividade na execução do código e consequentemente dos resultados obtidos, foi necessário proceder à instalação de ambiente gráfico na máquina, com o objetivo de aceder e alterar código de uma forma mais simples e rápida, para além de permitir a leitura e análise imediata de gráficos e *outputs*. Contudo, esta opção é acessível via remota, através de aplicações como *TeamViewer,* ou VNC. Implicando assim que o acesso adequado ao ambiente gráfico esteja dependente da rede *internet*. A Figura 34 demonstra o ambiente gráfico utilizado.

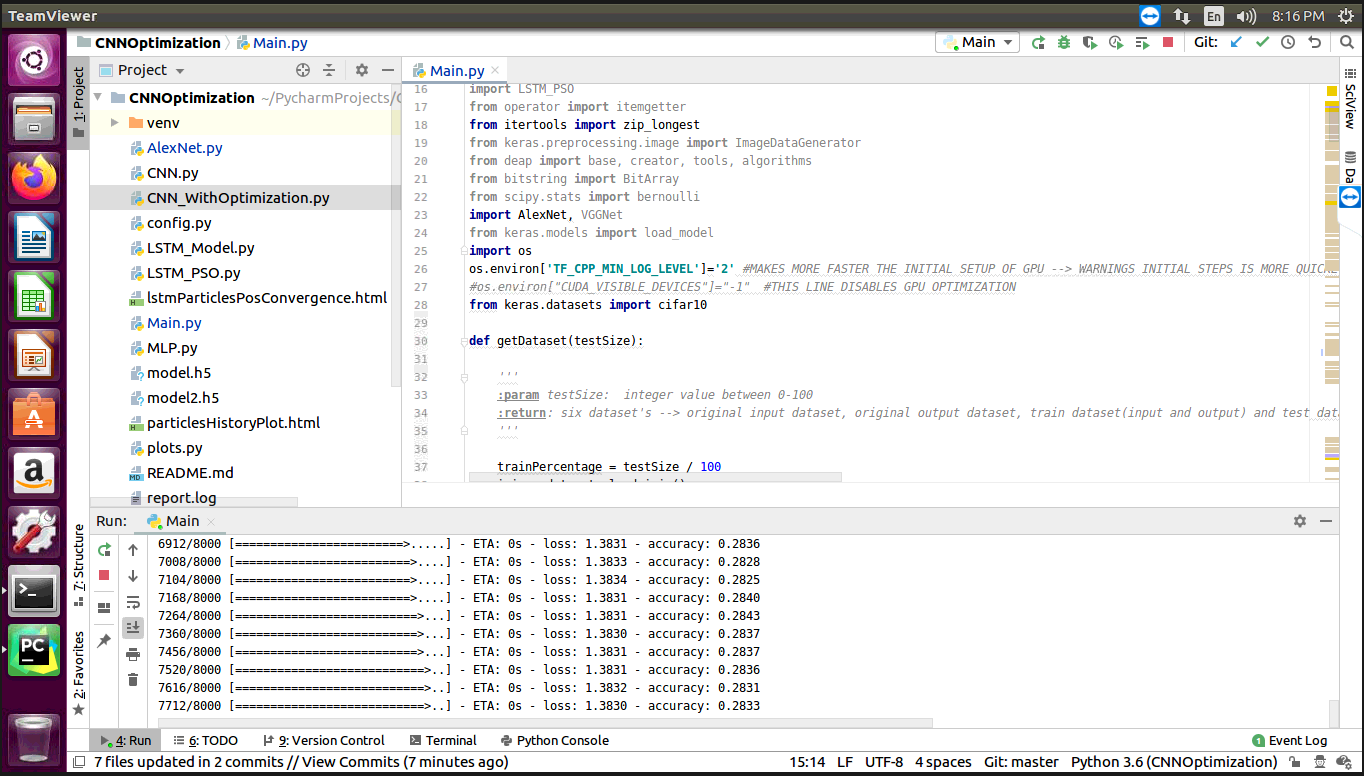


Figura - Ambiente Gráfico

**Otimização recorrendo ao algoritmo PSO**:

Dessa forma foi aplicado o PSO na tentativa de identificar os valores mais adequados para os vários parâmetros do modelo CNN, mais concretamente o nº de filtros e nº de neurónios, das camadas convolucionais e *dense layers* respetivamente.

A abordagem considerada foi a mesma já referida anteriormente (tópico: aplicação do PSO como otimizador do modelo CNN).

Para tal, foi considerado um nº de partículas igual a 15, um nº de iterações de 20 e por último um nº de *epochs* igual a 50. Desta forma, espera-se a obtenção de uma rede mais coesa e mais ajustada à função de custo a minimizar (já descrita atrás).

As Figuras 21 e 22 demonstram os resultados obtidos, da aplicação do PSO, respetivamente os valores referentes ao nº de filtros e de neurónios (dimensões do problema) e variação do custo ao longo das iterações.

Posteriormente, existiu a necessidade de verificar executar o modelo, considerando os valores indicados pelo PSO, de modo a que fosse possível comprovar a efetividade ou não da sua aplicação. As Figuras 23 e 24 ilustram respetivamente a variação do custo e da *accuracy* ao longo do tempo.

Comparando, os resultados obtidos nas Figuras 23 e 24 com os enumerados nas Figuras 5 e 6, é possível concluir que a utilização do algoritmo evolucionário *Particle Swarm Optimization*, permitiu a obtenção de melhores resultados. Visto que, existiu uma aprendizagem entre as várias partículas do problema, o que permitiu a convergência para uma “boa” solução. Por outro lado, a não utilização de técnicas de otimização na definição dos modelos, não permite a obtenção de um modelo aprimorado, sendo necessário recorrer a técnicas pouco práticas e dolorosas, como a típica técnica de “tentativa em erro”, ou a utilização de *grid search*.



****

****

****

**Otimização recorrendo ao algoritmo GA**:

O algoritmo evolucionário *Genetic Algorithm* já fora aplicado como otimizador de uma rede CNN, tal como fora descrito anteriormente. Contudo, e tal como explicado, o exemplo aplicado não considerou a utilização adequada de um nº de gerações, população e nº de *epochs* (fator ainda mais preponderante, na avaliação das soluções).

Sendo assim, foi aplicado o GA considerando um valor já razoável quer do nº de gerações, ppulação e nº de *epochs*, sendo estes 20, 20 e 50 respetivamente. Desta forma, a aplicação do GA torna-se mais robusta, implicando assim que os seus resultados à partida também o sejam. Ao contrário do PSO, o GA é um algoritmo mais aleatório, isto é, a adição de mais gerações nem sempre significa uma melhoria da aprendizagem e/ou dos resultados obtidos. Ao contrário do PSO, que ao considerar uma aprendizagem em grupo, permite que a probabilidade das partículas convergirem para melhores soluções seja maior. Ainda assim, é expectável que o GA apresente melhores resultados do que os fornecidos previamente, e que ainda permita a obtenção de valores adequados para os parâmetros do modelo, permitindo assim a definição de um modelo mais otimizado, e que ao fim de 50 *epochs* revele um melhor custo e *accuracy*, que a representação das Figuras 5 e 6.

Seguidamente, a Figura 35 representa os valores dos filtros e de neurónios, que apresentam um menor custo (considerando a função de custo, já referida anteriormente).

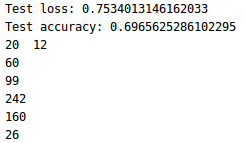


Figura - Resultados obtidos: (1 Linha) custo, (2) accuracy, (3) melhor resultado no 20º elemento e 12 geração, (4,5,6) valores dos filtros das camadas convolucionais utilizadas, (7,8) valores dos neurónios das Dense Layers

Da análise aos valores obtidos é possível concluir que os mesmos revelam-se mais expressivos (maiores), que os definidos na experiência realizada atrás, e expressa nas Figuras 5 e 6. Mas, é notório que o algoritmo seguiu o mesmo padrão definido na aplicação da rede *AlexNet* anterior, isto é, o nº de filtros e de neurónios vai aumentando à medida que a rede se revela mais profunda. À primeira vista os resultados finais obtidos ao fim de 50 *epochs* revelam-se condizentes com os obtidos anteriormente. Contudo, é importante aferir o comportamento do modelo considerando um maior nº de *epochs*.

Tal, como fora efetuado nos casos anteriores, seguiu-se a execução do modelo CNN, considerando os valores identificados pelo algoritmo GA, e ainda 150 *epochs*. As Figuras 36 e 37 ilustram respetivamente a variação do custo e da *accuracy*.

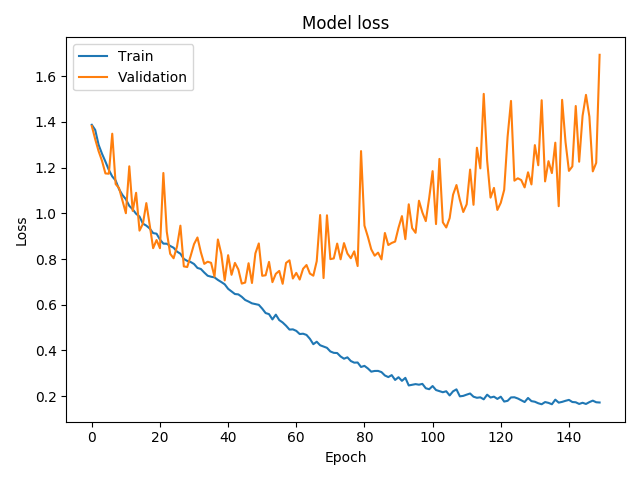


Figura - Variação do custo ao longo do tempo - Custo final de Teste = 1.70

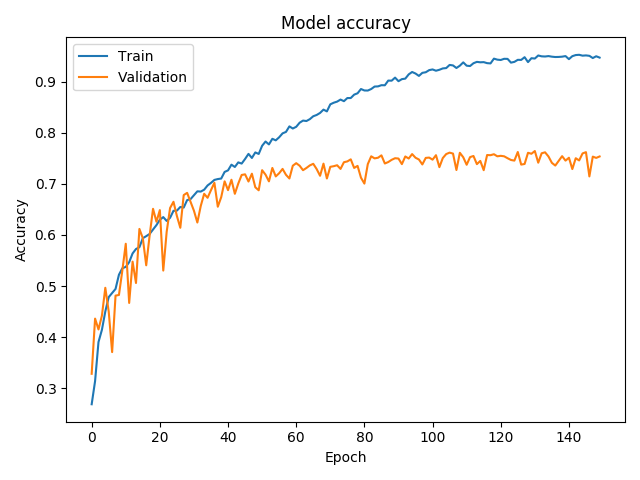


Figura - Variação da accuracy ao longo do tempo - Accuracy final de Teste = 75,3%

Como podemos verificar a aplicação do GA, proporcionou resultados adequados considerando as 50 *epochs* iniciais. Contudo, à medida que o nº de *epochs* vai aumentando, o modelo tende a sofrer de *overfitting*, ou seja, de modo a garantir um modelo adequado seria necessário considerar a aplicação do algoritmo de otimização considerando um valor mais elevado de *epochs,* de modo a que fosse possível atenuar e/ou evitar modelos que sofram um efeito tão exaustivo de *overfitting*. Mas, importa referir que o tempo de execução seria ainda mais “doloroso”, visto que considerando os valores descritos atrás, 20 indíviduos e gerações e 50 *epochs*, o algoritmo demorou 1 dia (entre 20-24 horas) a executar.

Ou seja, é importante garantir que antes de aplicar o GA, exista um estudo prévio de qual a arquitetura que melhor se adequa ao problema, e ainda qual o *range* de valores mais adequados para um determinado hiperparâmetro, de modo a que seja possível reduzir o tempo e custo computacional exigido na aplicação dos algoritmos.

A aplicação do conceito de *Data Augmentation,* também deve ser considerada após a aplicação dos algoritmos, com o intuito de melhorar a performance do modelo obtido.

Outra das abordagens passa pela otimização progressiva dos modelos. Isto é, deve-se considerar os limites das dimensões do problema baixos, numa fase inicial, sendo que progressivamente estes valores devem ser aumentados, permitindo assim identificar quais os *ranges* mais indicados para um determinado modelo. Esta abordagem permite reduzir os custos de aplicação dos algoritmos de otimização, ajudando ainda a identificar e evitar problemas de *overfitting* e resultados tendenciosos.

Da aplicação dos algoritmos de otimização foi ainda possível concluir que a função de custo deveria ser otimizada e reformulada. Ou seja, a função de custo atual não têm em consideração a margem de custo e *accuracy,* entre a fase de treino ou de validação, tendo apenas em consideração a *accuracy* final de teste obtida. Esta não consideração, resulta em modelos, que podem sofrer de *overfitting,* tal como o modelo obtido e descrito nas Figuras 26 e 27. Sendo assim, é importante minimizar as diferenças de custo e de *accuracy* obtidas ao longo das várias *epochs* aplicadas, pois dessa forma é possível obter um modelo generalista.

Referências:

[1] <https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029>

[2] <https://www.researchgate.net/publication/321823631_A_Particle_Swarm_Optimization-based_Flexible_Convolutional_Auto-Encoder_for_Image_Classification>

[3] <https://arxiv.org/ftp/arxiv/papers/2001/2001.05670.pdf>

[4] <https://www.biorxiv.org/content/biorxiv/early/2018/07/11/364323.full.pdf>

[5] <https://arxiv.org/ftp/arxiv/papers/1906/1906.07794.pdf>

**Convolutional neural network models for cancer type prediction based on gene expression[5]:**

**Dataset**: RNA-Seq do TCGA

**Feature Selection**: 𝑙𝑜𝑔2 (𝐹𝑃𝐾𝑀 + 1), onde FPKM é o nº de fragmentos por kilobase por milhões de leituras; Genes com baixo mean < 0.5 or st. dev. < 0.8, foram eliminados; Eliminando assim features que representam pouca informação, ou ruído; No total sobraram ainda 7100 genes; O artigo [9] salienta outra abordagem utilizada no processo de *feature selecion,* mas que não se revela tão fácil de entender;

**Ordenação dos Genes**: É importante ordenar corretamente os genes, de acordo com as suas correlações. Um dos métodos muito conhecidos é a ordenação pela posição dos seus cromossomas; O estudo realizado em [4] também evidenciou a importância da ordenação dos genes, de acordo com o nº do cromossoma, visto que cromossomas adjacentes apresentam uma maior probabilidade de “interagirem” entre si;

**O estudo realizado em [5] refere que contrariamente aos problemas de classificação de imagens, que normalmente requerem redes CNN mais profundas, problemas relacionados com a predição biomédica revelam melhores resultados, quando são aplicadas camadas CNN mais simples;**

Relativamente à representação das redes CNN, para resolução destes problemas, são várias as abordagens que podem ser utilizadas. Como por exemplo, recorrer à utilização de redes convolucionais 1D, sendo que o modelo recebe como *input* um vetor, que contém a expressão dos genes que compõem o problema. Estas redes são muito utilizadas na resolução de problemas, que envolvam alguma dependência entre as *features (features* próximas). Caso, não exista uma correlação entre as *features* do problema, normalmente o valor do *stride* é igual ao tamanho do *kernel.* Dessa forma, é capturada apenas a informação global, inserida no *kernel,* e não se avaliam dependências entre as *features*;

Outra das abordagens mais comuns para a resolução do problema consiste na transformação do *input,* de um espaço unidimensional (vetor) para um espaço 2D. Os estudos [4] e [5] demonstram exemplos desta abordagem;

Por último, outra das abordagens que pode ser utilizada na resolução deste problema, incide na aplicação “customizada”, de ambas as técnicas descritas anteriormente. Ou seja, é considerada a utilização de um *input 2D,* mas ao invés de ser utilizado um *kernel 2D* na filtragem convoluvional, recorre-se à utilização de um *kernel* 1D. Esta abordagem, em contextos específicos permite uma identificação mais rigorosa de características/padrões nos dados de entrada;

**Dataset’s**:

Estudo muito interessante <https://jcheminf.biomedcentral.com/articles/10.1186/s13321-019-0364-5> --> recorre à utilização de AlexNet, Resnet, VGGNet 🡪 na resolução de um problema inserido no mesmo âmbito;

Gene Expression Dataset’s 🡪 Time Series:

[1] <https://www.cs.cmu.edu/~jernst/st/>

[2] <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE124824>:   
RNA-seq in 247 B-lineage ALL

[3] <https://www.ncbi.nlm.nih.gov/pubmed/15718313> --> Dataset já utilizado: *Breast Cancer*

[4] <https://www.ncbi.nlm.nih.gov/pubmed/17999412> --> <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE5847> : Tumor and stroma from breast by LCM

[5] <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE14426> 🡪 Pancreatic Cancer

[6] <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi> --> Colon Cancer

[7] <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE5462> 🡪 Breast Cancer

[8] <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi> --> Alzheimer poucas amostras

[9] <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=gse48046> : Gene expression analysis of Early immature and Late mature T-ALL cell lines

[10] <https://www.ncbi.nlm.nih.gov/sites/GDSbrowser?acc=GDS4794> 🡪 Small cell lung cancers

[10] <https://hgserver1.amc.nl/cgi-bin/r2/main.cgi>

Classificação de Imagens Biomédicas:

[1] <http://www.eng.usf.edu/cvprg/Mammography/Database.html>

[2] <https://warwick.ac.uk/fac/sci/dcs/research/tia/data/crchistolabelednucleihe/>

[3] <https://github.com/sfikas/medical-imaging-datasets>

- DermNet - Skin disease atlas (23 image classes and 23,000 images);

- <https://www.kaggle.com/c/intel-mobileodt-cervical-cancer-screening/data>;

-<http://www.ia.unc.edu/MSseg/;> 🡪 Challenge

[4] <https://github.com/beamandrew/medical-data>

- Isic Archive - Melanoma;

- INbreast – São João (Hospital Mamografia); - <http://medicalresearch.inescporto.pt/breastresearch/index.php/Get_INbreast_Database>;

- UCI Dataset’s: Parkinson, Tiroide, etc;

[5] <https://wiki.cancerimagingarchive.net/display/Public/CBIS-DDSM> ou parcelar --> <http://www.eng.usf.edu/cvprg/Mammography/Database.html>

[6] <https://wiki.cancerimagingarchive.net/display/Public/Wiki>

[7] <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE30> 🡪 Multiplex three dimensional brain gene expression mapping in a mouse model of Parkinson's disease