UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO CIENTÍFICA
DEPARTAMENTO DE MATEMÁTICA APLICADA

JOSÉ GUILHERME RIZZO DE BARROS

Um estudo sobre redes neurais fractais

Campinas
11/01/2021

JOSÉ GUILHERME RIZZO DE BARROS

Um estudo sobre redes neurais fractais

Monografia apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos para obtenção de créditos na disciplina Projeto Supervisionado, sob a orientação do(a) Prof. João Batista Florindo.

Resumo

Este trabalho traz um estudo sobre uma promissora e recente arquitetura em relação às de redes neurais artificiais. Tal arquitetura faz uso da propriedade autosimilar presente em objetos fractais para construir a configuração da rede, dando origem ao nome e estabelecendo diferentes níveis de complexidade. O trabalho conta com uma breve construção histórica das redes neurais e emprega a arquitetura fractal utilizando camadas convolucionais. Além disso, a implementação realizada em Keras com Tensorflow como *backend* é comparada com uma rede neural residual (ResNet50), utilizando-as para resolver um problema de classificação binária (de pequeno porte) e um problema multiclasses. Será possível notar que os resultados iniciais da rede neural fractal mostram-se competitivos e em alguns casos até superiores aos obtidos com o uso da ResNet, trazendo consigo algumas vantagens e inovações em termos de implementação, tais como novas técnicas de regularização, boa performance sem aumento artificial de dados, capacidade de supervisão do processo em redes profundas, semelhante ao aprendizado no paradigma estudante-professor, dentre outros. Por fim, aponta-se a necessidade da realização de testes mais robustos e que explorem ao máximo a capacidade da arquitetura, para que seja possível ampliar o campo de estudo sobre o tema.

Abstract

This report brings a study on a promising and recent architecture of artificial neural network. This architecture uses the self-similar property present in fractal objects to build the network model, giving rise to the name "fractal"and establishing different levels of complexity. This report starts with a brief historical construction of neural networks and then employs fractal architecture using convolutional layers. In addition, an implementation using the Python framework Keras, with Tensorflow as backend, is compared with a residual neural network (ResNet50), and is used to solve a binary classification problem (small size) and a multiclass problem. It is possible to notice that the initial results of the fractal neural network are competitive and in some cases even superior to those obtained with the use of ResNet, bringing some advantages and innovations, such as new regularization techniques, good performance without artificial data augmentation, capacity of supervision of the process in deep networks, similar to the learning paradigm student-teacher, among others. Finally, we observe the need for more robust tests that could fully use all the potential of the architecture, possibly extending the field of study on the subject.

Conteúdo

1 Introdução

Atualmente, há um grande desenvolvimento de pesquisas relacionadas ao estudo de tópicos envolvendo inteligência artificial. Dentro deste contexto, segundo Li Deng [2014] e Goodfellow et al. [2016], destaca-se a área destinada a desenvolver projetos utilizando *Deep Learning* como ferramenta, principalmente quando o interesse é desenvolver algoritmos que reconheçam padrões em imagens. Diante desta problemática, a principal técnica implementada para abordar problemas que envolvam imagens na base de dados trata da utilização de redes neurais que sejam capazes de aprender e distinguir certos padrões presentes nos dados de entrada. Com isso, ao longo da história diversas abordagens foram realizadas para a construção dessas arquiteturas, evoluindo de modelos mais robustos aos mais sofisticados e utilizados hoje em dia. Como exemplo, destacam-se as redes neurais convolucionais, com maior popularidade em trabalhos envolvendo base de imagens, pois sua implementação extrai diretamente as informações mais importantes da imagem, utilizando um número significativamente menor de parâmetros em comparação com técnicas anteriores e robustas (como redes neurais completamente conectadas). Por essa razão, este projeto busca mesclar a técnica de implementação de redes neurais convolucionais sob a ótica de uma recente arquitetura, dando origem a uma rede neural conhecida como rede neural fractal. Portanto, a partir de Larsson et al. [2017], este relatório trará as principais características desta arquitetura, bem como resultados de testes comparativos de pequeno porte realizados para compreender melhor o funcionamento da rede. Além disso, comentários sobre os principais desafios encontrados serão feitos ao longo do texto.

2 Um breve histórico

Nos últimos anos, a implementação de redes neurais para a resolução de problemas de classificação sofreu uma ascensão que impulsionou o desenvolvimento de diversas pesquisas relacionadas a área. No entanto, embora tópicos relacionados à machine learning de maneira geral, tenham ganhado muita evidência no âmbito acadêmico e no mercado de trabalho, a origem das redes neurais não é recente e não se restringe à problemas de classificação, tendo suas primeiras inspirações na década de 40, quando o neurofisiolo-

gista Warren McCulloch e o matemático Walter Pitts escreveram um artigo expondo um modelo de funcionamento de um neurônio cerebral. Entretanto, além do próprio modelo proposto, o que ganhou destaque foi o fato de não apenas descreverem seu funcionamento, como também simularem uma simples rede neural utilizando circuitos elétricos. Com isso, mesmo tendo nascido em 1943, somente em 1950 é que foi possível começar a implementar simulações de redes neurais um pouco mais complexas devido ao avanço dos computadores. Com isso, a primeira tentativa de implementar estas simulações foi de Nathanial Rochester nos laboratórios de pesquisa da IBM, que por sua vez, não obteve sucesso. Com o passar dos anos, grandes acontecimentos à respeito do tema tiveram destaque. Como exemplo, destacam-se os trabalhos realizados por Bernard Widrow e Marcian Hoff, que desenvolveram dois modelos conhecidos como ADALINE e MADALINE, sendo o primeiro utilizado para reconhecer padrões em sequências de bits, desenvolvendo a habilidade de predizer o valor do próximo bit, e MADALINE sendo a primeira implementação de uma rede neural aplicada em um problema do mundo real. Em específico, MADALINE utiliza um filtro adaptativo que elimina ecos em linhas telefônicas e possui uso comercial até hoje, embora antigo *.

Outros grandes trabalhos impulsionaram o avanço da estrutura formadora das redes neurais e dos própios neurônios e suas formas de ativação. O modelo básico inicial proposto por McCulloch e Pitts considerava uma entrada discreta binária (cada neurônio da entrada assumia valor 0 ou 1) e a partir da soma dessas entradas, uma saída também binária era definida com base em um valor limiar. Posteriormente, houve a criação da unidade denominada perceptron, que é uma adaptação da rede de McCulloch e Pitts, permitindo agora que as entradas possuam valores pertencentes ao conjunto dos números Reais. Além disso, são atribuídos pesos às conexões entre neurônios de entrada e saída, que por sua vez ainda é binária e definida com base no somatório dos pesos multiplicados por suas respectivas entradas, tendo ainda um limiar para a tomada de decisão. Por fim, o modelo de neurônio que tornou-se mais popular é o modelo logístico, que adapta a saída para valores reais entre 0 e 1 e possui uma função que opera sobre os elementos da entrada para definir a saída da rede, denominada função de ativação. Abaixo seguem imagens que

*https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html

exemplificam os modelos descritos anteriormente:

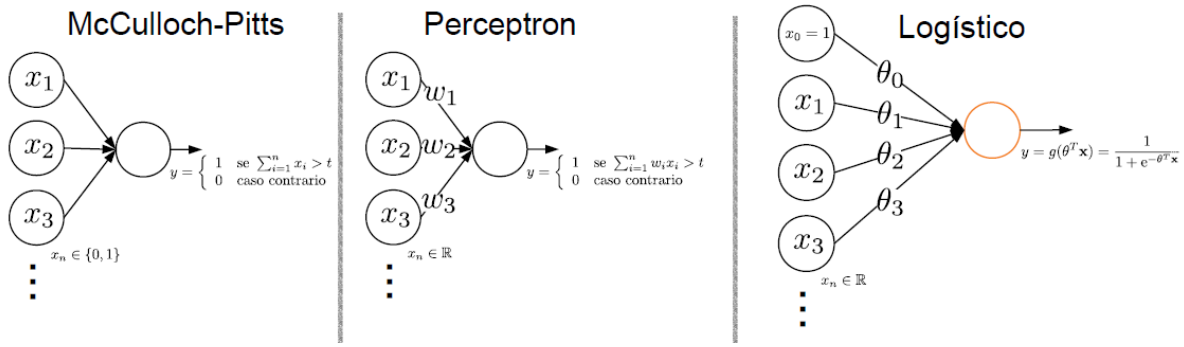

Figura 1: Modelos visuais de neurônios em redes neurais.

Note que na imagem acima, g representa a função de ativação, θ representa os pesos e θ_0 recebe o nome de *bias*, que é um valor constante que ajuda no ajuste dos pesos da rede. Existem diversas funções de ativação e a descrita na imagem recebe o nome de sigmóide, sendo muito popular durante alguns anos. Porém, atualmente faz-se uso da função Relu (max(0,x)), escolhida também para a implementação do código da arquitetura estudada. A seguir, serão comentadas as características das Redes Neurais Convolucionais, desde uma abordagem histórica até um comparativo com as redes completamente conectadas, que nada mais são do que ligações por camadas entre diversos neurônios logísticos.

3 Redes Neurais Convolucionais

Redes neurais convolucionais definem um método para reconhecimento de padrões em imagens que inspira-se na visão humana, pois é construído com base na definição de filtros que são responsáveis por aprender informações diferentes a partir de uma mesma imagem "observada". Assim, para entender como os filtros atuam nas imagens, vale comentar que uma imagem pode ser representada por uma matriz de pixels, em que cada elemento da matriz representa o valor número associado ao pixel correspondente. Estes valores numéricos representam a intensidade de cores sobre uma determinada região da imagem. Para imagens definidas em preto e branco, ou seja, em escala de cinza, há apenas uma matriz de pixels associada cujo os valores variam entre 0 e 255, no qual 0 representa a cor preta e 255 a intensidade mais branca. Todos os outros valores entre 0 e

255 são tonalidades de cinza que compõe a imagem.

Em geral, a maioria das imagens existentes nos dias atuais são coloridas. Assim, para o formato RGB por exemplo, não há apenas uma matriz de pixels relacionada à cada imagem e sim três matrizes que descrevem as intensidades de cada cor que compõe a imagem (Red - Green - Blue). Redes convolucionais podem trabalhar com imagens em escala de cinza ou coloridas, com a diferença de que no caso das imagens coloridas, como há mais de uma matriz associada à cada imagem, é como se os filtros operassem em cada matriz de uma imagem reconhecendo padrões e extraindo importantes informações de maneira independente. Para fins didáticos, abaixo será ilustrado a forma como a convolução ocorre entre um determinado filtro e uma matriz.

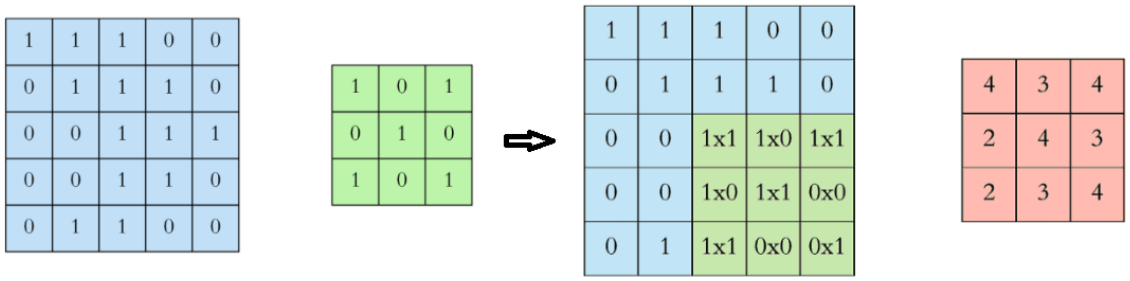

Figura 2: Exemplo de funcionamento de uma convolução. Fonte: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

A imagem acima descreve em azul o que seria a matriz de pixels da imagem e em verde a matriz que faz o papel do filtro, também conhecido como kernel. Do lado direito, há a representação do resultado final pós convolução, obtendo a matriz de cor rosa que possui um tamanho menor do que a matriz original. Esta é uma das principais vantagens do uso das redes neurais convolucionais. O processamento dos dados de entrada pela rede reduz o tamanho das imagens originais, tornando as operações subsequentes mais rápidas e reduzindo o número de parâmetros conforme o avanço do processo.

A operação de convolução, que matematicamente mede a soma do produto de duas funções dadas, no contexto de redes neurais traz consigo outras técnicas importantes que agregam na forma como a imagem é percorrida. Neste sentido, há dois conceitos importantes denominados *padding* e *stride*, sendo o primeiro responsável por definir como

a convolução opera nas bordas, já que se habilitado, o processo de *padding* cria uma espécie de borda artificial na imagem, preenchida com zero, um ou qualquer outro valor conveniente ao processo. Por outro lado, o *stride* tem o papel de definir como o kernel percorre a matriz de pixels da imagem, ou seja, o passo com que percorre-se a matriz de entrada. Assim, caso o *stride* seja maior que um, por exemplo, o resultado da convolução terá dimensão menor que o caso em que o passo é igual a um. A seguir, a imagem abaixo ilustra como o processo de *padding* atua no processo de convolução:

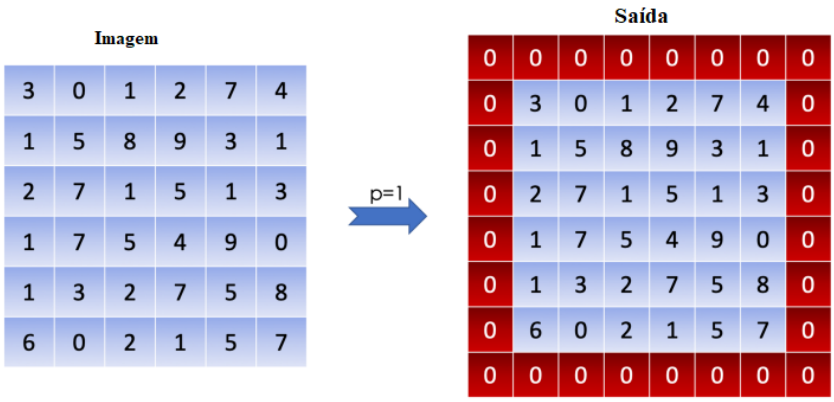

Figura 3: Exemplo de funcionamento do processo de *padding*. Adaptado de: https://towardsdatascience.com/convolutional-neural-networks-mathematics-1beb3e6447c0

Observe que existe um parâmetro $p = 1$. Este, por sua vez, define o número de elementos pertencentes à cada um dos quatro cantos da imagem, ou seja, determina a "espessura"da borda. O conceito de *stride* será esquematizado no exemplo sobre *pooling*. Matematicamente, é possível também definir o conceito de convolução e dos conceitos associados à ela até aqui. Considere que uma imagem possa, em geral, ser representada por um tensor com as seguintes dimensões:

$$dim(imagem) = (n_H, n_W, n_C)$$

Com n_H representando o tamanho da altura ($Height$), n_W a largura ($Widht$) e n_C o número de canais. Da mesma maneira, define-se a dimensão do filtro como (f, f, n_C), em que f determina o tamanho do filtro e n_C é o número de canais da imagem, ou seja, aplica-se o filtro à cada canal para compor a saída. Portanto, a operação de convolução pode ser representada de forma multi-dimensional pela relação:

$$Conv(I,K)_{x,y} = \sum_{i=1}^{f}\sum_{j=1}^{f}\sum_{k=1}^{n_C} K_{i,j,k} I_{x+i-1,y+j-1,k}$$

Em que I denota a imagem e K o filtro utilizado e os sub-índices x, y denotam a posição do elemento na matriz de convolução. Com isso, além de ser possível estabelecer de maneira genérica a convolução em uma certa camada da rede, é também possível definir a dimensão da convolução, como $\left(min\_int\left[\frac{n_H+2p-f}{s}+1\right], min\_int\left[\frac{n_W+2p-f}{s}+1\right]\right)$ se $s > 0$, em que $min\_int[x]$ representa o maior inteiro menor que x. Caso $s = 0$, a dimensão é definida por $(n_H + 2p - f + 1, n_W + 2p - f + 1)$. Mais precisamente, considere que na *l-ésima* camada, denota-se:

- Na entrada: $a^{[l-1]}$ com tamanho $(n_H^{[l-1]}, n_W^{[l-1]}, n_C^{[l-1]}$. Note que $a^{[0]}$ representa a imagem inicial do problema.
- $Padding = p^{[l]}$, $stride = s^{[l]}$.
- Número de filtros: $n_C^{[l]}$ em que cada filtro $K^{(n)}$ tem dimensão $(f^{[l]}, f^{[l]}, n_C^{[l-1]})$.
- *Bias* da n-ésima convolução: $b_n^{[l]}$.
- Função de ativação: $\psi^{[l]}$.
- Saída $a^{[l]}$ com tamanho $(n_H^{[l]}, n_W^{[l]}, n_C^{[l]})$.

Então, a convolução entre a entrada da camada l e o *n-ésimo* filtro presente na configuração da rede é dada por:

$$Conv(a^{[l-1]}, K^{(n)})_{x,y} = \psi^{[l]}\left(\sum_{i=1}^{f^{[l]}}\sum_{j=1}^{f^{[l]}}\sum_{k=1}^{n_C^{[l-1]}} K_{i,j,k}^{(n)} a_{x+i-1,y+j-1,k}^{[l-1]} + b_n^{[l]}\right)$$

Com $dim(Conv(a^{[l-1]}, K^{(n)})) = (n_H^{[l]}, n_W^{[l]})$. Essa operação é realizada para cada $n \in [1, 2, ..., n_C^{[l]}]$ e desta maneira, a *l-ésima* entrada $a^{[l]}$ pode ser obtida aplicando a função de ativação sobre cada combinação entre a entrada $a^{[l-1]}$ e os filtros existentes. Com isso, é possível concluir que $dim(a^{[l]}) = (n_H^{[l]}, n_W^{[l]}, n_C^{[l]})$ e que os parâmetros aprendidos nesta

camada resultam em ($f^{[l]}$ x $f^{[l]}$ x $n_C^{[l-1]}$) x $n_C^{[l]}$ (provenientes dos filtros) e (1 x 1 x 1) x $n_C^{[l]}$ (provenientes do *Bias*).

Por fim, à respeito da implementação de redes neurais convolucionais, há ainda alguns conceitos finais pelo qual o conjunto de treinamento deve passar. O primeiro deles diz respeito à camada de *Pooling*, que tem como objetivo atuar sobre a matriz resultante da operação de convolução com o objetivo de reduzir a dimensionalidade do problema e assim, o tamanho da imagem. Essa redução também implica numa redução do número de parâmetros treinados pela rede. Com esta técnica, ocorre também uma diminuição da variância a pequenas alterações. Segue abaixo uma representação visual desta operação:

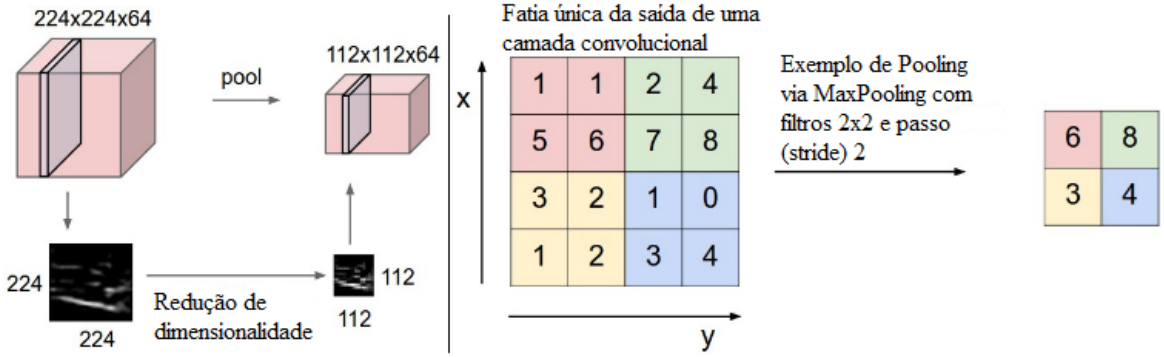

Figura 4: Exemplo de funcionamento de uma camada de *Pooling*. Adaptado de: https://cs231n.github.io/convolutional-networksl

Na imagem acima, é possível notar que o exemplo utiliza um filtro de tamanho 2x2 para realizar o processo de *Pooling*, ou seja, a matriz original 4x4 tem seus elementos percorridos por uma sub-região de tamanho 2x2 e *stride* igual a 2. Dessa forma, o resultado do processo é uma matriz 2x2 com seus elementos sendo o máximo dentre os elementos de cada sub-região percorrida. Esse processo configura uma técnica de *Pooling* chamada *MaxPooling*. Existem outras formas de realizar o *Pooling*, tomando a média dos elementos da sub-região ou a soma de seus elementos.

Assim como no caso da convolução, a dimensão pós camada de *pooling* é definida de maneira semelhante, exceto pelo fato de que obtém-se um vetor tridimensional como resposta, tendo as duas primeiras coordenadas idênticas ao caso convolucional men-

cionado acima e na terceira coordenada, o valor correspondente à n_C.

Além do *Pooling*, há também uma técnica conhecida como *Dropout*, que tem como objetivo desativar certas áreas da rede para que nenhuma região ou conjunto de parâmetros tornem-se mais responsáveis pelo aprendizado do processo. Com isso, ganha-se uma forma de regularização sem o uso de modificações da função de custo, ou seja, diminui-se a probabilidade de *overfitting* (sobreajuste dos dados, ou seja, um bom desempenho no treinamento e fraca performance na etapa de validação) da rede. Por fim, há dois processos pelo qual os dados de entrada passam para obter a saída da rede, que são conhecidos como *flatten* e *dense layer* (camada densa). Na etapa de *flatten*, como o próprio nome sugere, ocorre uma transformação da matriz resultado das operações de convolução e *pooling*, transformando-a num vetor unidimensional. Em seguida, a camada densa recebe como entrada o resultado da etapa de flatten e é composta por neurônios logísticos que calculam os pesos e realizam a predição sobre a saída.

As redes neurais convolucionais originaram-se em 1980, implementadas por Yann LeCun, pesquisador pós-doutor em Ciência da Computação, hoje considerado um dos "pais"da teoria de *deep learning*. Entretanto, o ganho de popularidade das redes convolucionais ocorreu em 2012, quando a rede AlexNet ganhou o desafio ImageNet, tendo como base em sua construção as redes convolucionais. Neste projeto, utilizou-se uma abordagem baseada em redes neurais convolucionais para construir a arquitetura fractal. A seguir, após esta breve explanação sobre os conceitos base para a construção da rede, haverá a exposição da teoria sobre rede neural fractal.

4 Rede Neural Fractal

Antes de descrever a composição da rede fractal, vale comentar sobre o porquê do nome dado à essa arquitetura. Primeiramente, um fractal pode ser entendido como um objeto geométrico que goza da propriedade de autosimilaridade, ou seja, cada região do objeto em que se possa dar *zoom* assemelha-se ao objeto original. A partir desta noção é que constrói-se a rede neural fractal, explorando o conceito de autosimilaridade e fazendo com que a partir de uma regra de expansão simples, possa ser possível compor estruturas emergentes e redes neurais profundas.

Atualmente, uma das redes neurais mais populares e utilizadas em classificação de imagens é conhecida como rede neural residual ou simplesmente ResNet. Tal arquitetura possui propriedades interessantes que são adaptadas para o caso das redes neurais fractais e garantem resultados competitivos quando comparados. A ResNet é uma rede que possui como caracterísitca a implementação de uma técnica conhecida como *skip connections* (salto de conexões). Esta estratégia faz com que determinadas entradas não sejam processadas por grupos de camadas convolucionais, dando esta noção de salto entre as conexões. Isto faz com que o problema de desaparecimento do gradiente no processo de retropropagação da rede seja atenuado, pois soma-se a entrada não processada às informações processadas pelas camadas antes de gerar um novo sinal a ser transmitido para as etapas subsequentes. O processo é esquematizado abaixo:

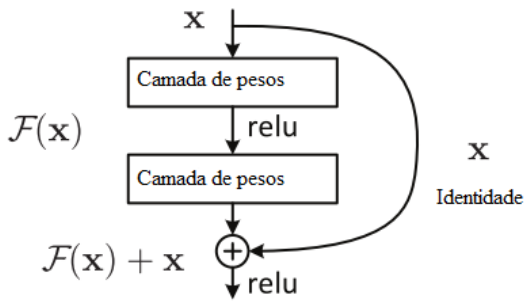

Figura 5: Exemplo de funcionamento da ResNet. Adaptado de: https//towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035l

Esta característica presente nas redes residuais faz com que seja possível trabalhar com redes neurais extramamente profundas, visto que nem toda a informação é perdida no processamento e é utilizada para a entrada de camadas subsequentes às puladas na etapa anterior. Com isso, neste projeto, além dos testes realizados sobre o código para a rede neural fractal, também foi implementado a ResNet50 para fins de comparação de desempenho. Maiores informações sobre isto estarão na seção **Resultados**.

Agora, em posse das informações referentes às diferentes arquiteturas presentes em redes neurais, desde o uso de neurônios logísticos, passando pela implementação utilizando camadas convolucionais e posteriormente utilizando estas técnicas base para implementar arquiteturas que manipulem a forma como o conjunto de dados de treinamento interage com tais conceitos iniciais, é possível fazer uso da propriedade autosimilar

presente nas redes neurais fractais para construir estruturas complexas a partir de uma regra razoavelmente simples de expansão. A figura abaixo exemplifica o método de construção:

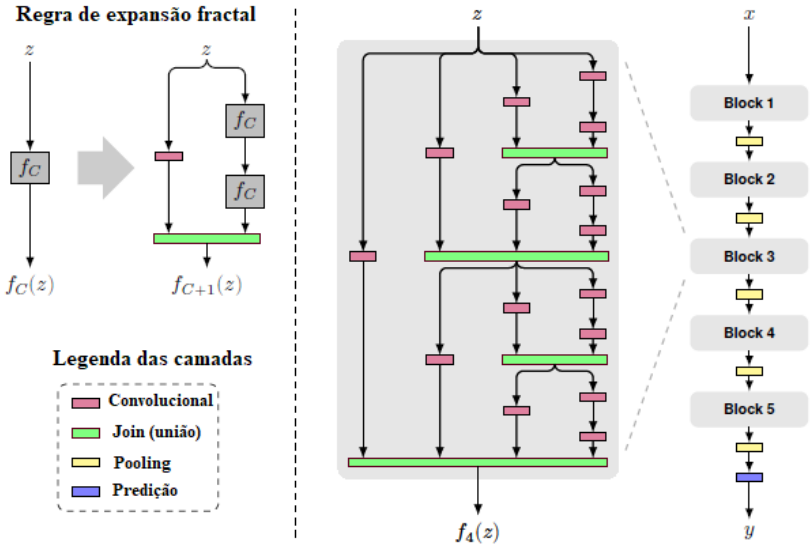

Figura 6: Exemplificação da rede neural fractal. Adaptado de: Larsson et al. [2017]

Como a imagem acima mostra, a arquitetura de exemplo é composta por cinco blocos, sendo que ao final do processamento em um determinado bloco há uma camada de *pooling* antes de inicializar a operação no bloco seguinte. Note que do lado esquerdo da figura, há a representação da regra de expansão da rede, em que $f_C(z)$, com C >1, denota a aplicação da regra sobre $f_{C-1}(z)$, de acordo com a regra de expansão utilizada. No caso, $f_1(z)$ é uma simples camada convolucional. Com isso, é possível confirmar visualmente a estrutura autosimilar da rede e além disso, definir o conceito de profundidade como sendo o produto do número de blocos (B) por 2^{C-1}, em que C denota o número de colunas internas aos blocos. Note que 2^{C-1} resulta na quantidade de camadas convolucionais da última coluna do bloco (da esquerda para direita), que também é o caminho pelo qual há maior quantidade de camadas e portanto, maior processamento a ser feito. Assim, por ele há um ganho de profundidade pela rede. Matematicamente, segundo Larsson et al. [2017], a relação de construção da arquitetura pode ser obtida de maneira recursiva, enxergando o processo através de composições, da seguinte maneira:

Para uma rede em que $f_1(z) = conv(z)$ em que $conv(z)$ denota uma camada convolucional sobre a entrada, obtém-se que

$$f_{C+1}(z) = [(f_C \circ f_C)(z)] \oplus [conv(z)]$$

Em que $\circ$ denota composição e $\oplus$ a operação de *join*, definida logo abaixo.

Ainda sobre a questão da profundidade da rede, observe que ao descrever o conceito, utilizou-se como base uma das colunas de um bloco para descrever o fenômeno. Isto ocorre de maneira proposital, pois outro ganho da implementação desta arquitetura fractal segundo Larsson et al. [2017] é a possibilidade de obter bom desempenho considerando caminhos diferentes que ligam a entrada à saída de um bloco, recebendo o nome de sub-caminhos (na tradução livre). Entretanto, antes de comentar mais sobre estes sub-caminhos, observa-se que na figura [6] há em verde a operação *join*, que é uma técnica utilizada para unir as informações obtidas pelos diferentes caminhos que executam o processamento sobre uma mesma entrada. Na proposta dos autores, esta união é feita através da média de seus elementos.

Os sub-caminhos da rede podem ser vistos e entendidos melhor quando conhecidas as técnicas de regularização criadas pelos autores, que imitam o *dropout* e recebem o nome de *global_path* e *local_path*. Basicamente, a ideia em ambos os casos é desligar conexões da rede de forma a garantir independência no processo de aprendizado e com isso, garantir também que os sub-caminhos tenham bom desempenho (embora percam em acurácia quando comparados à rede completa) ao fornecer respostas mais rápidas em comparação à rede toda. Dito isto, é importante também estabelecer a diferença entre ambos: o *global_path* é uma técnica que desliga praticamente todas as conexões dentro de um bloco, mantendo apenas um sub-caminho ligando a entrada do bloco à saída do mesmo, enquanto que o *local_path* desabilita conexões aleatórias, mantendo um caminho que conecta entrada e saída do bloco, porém, podendo haver mais de um caminho realizando tal função. Ambas as técnicas constituem algo chamado de *drop_path*. A figura abaixo exemplifica o que foi dito anteriormente:

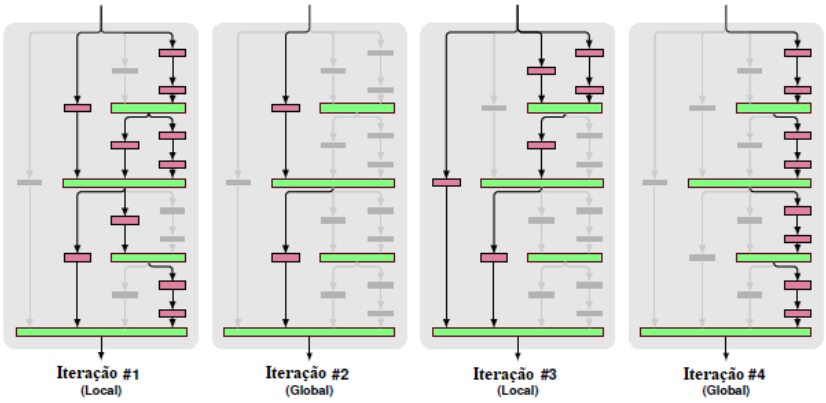

Figura 7: Exemplos de global_path e local_path. Adaptado de: Larsson et al. [2017]

Portanto, outra forma de compreender como ocorre o funcionamento do *local_path* e do *global_path* é observar que no segundo caso, não há possibilidade da rede executar o processo de *join* entre as colunas, sendo este permitido no caso do *local_path*. Por fim, segundo Larsson et al. [2017], é possível estabelecer alguns resultados interessantes obtidos através dos testes realizados pela equipe de pesquisa. Dentre eles, destacam-se:

1. O uso da rede fractal demonstrou que não é necessário apenas o uso de redes residuais para treinamento de redes profundas.

2. A regularização via *Drop-path* implica que a rede força com que cada conexão aprenda de forma individual em relação às demais.

3. Experimentos mostraram que não é necessário o uso de *Data Augmentation* (aumento artificial dos dados) para um melhor desempenho da rede neural fractal.

Além dos tópicos acima, de acordo com Larsson et al. [2017], é posível supervisionar a performance em sub-caminhos profundos e as técnicas utilizadas possibilitam que o aprendizado da rede ocorra segundo o método estudante-professor, em que as camadas mais rasas da rede são diretamente influenciadas pelas camadas de alta profundidade. E pela forma que a rede é contruída e as informações processadas, faz sentido que isso ocorra, pois através da operação de *join*, informações presentes nas camadas que fazem parte de sub-caminhos profundos não são completamente perdidas, visto que influenciam na atualização dos parâmetros das camadas mais rasas e isso pode justificar esta dinâmica interna da rede.

5 Resultados

Nesta seção, haverá a apresentação dos resultados obtidos através dos testes iniciais desta nova arquitetura, implementada com Keras* e Tensorflow como *backend*. Entretanto, vale comentar que os autores do projeto utilizaram uma linguagem chamada Caffe em conjunto com uma macro-linguagem denominada Crox para realizar a implementação da rede. Além disso, os testes feitos pela equipe de pesquisa foram baseados em redes com elevado grau de profundidade e com bases de treinamento com um enorme conjunto de dados. Portanto, diante do cenário global atual, os testes realizados neste projeto tiveram que se adaptar à capacidade computacional existente, não sendo possível utilizar os laboratórios de pesquisa da UNICAMP. Com isso, algumas alternativas foram criadas, dentre elas:

1. Testar o funcionamento da rede neural fractal em um problema de classificação binária e comparar com o desempenho da ResNet50.

2. Como a máquina utilizada não possui placa de vídeo e havia conflito interno com versões do python e das bibliotecas utilizadas, optou-se por utilizar o ambiente *Google Colaboratory* para a realização dos testes.

3. Diminuiu-se a profundidade da rede, visando apenas acompanhar seu desempenho em casos menores se comparados ao feito no artigo.

4. Diminuiu-se o número de épocas, para tornar os testes mais rápidos e verificar o comportamento inicial da arquitetura.

Com isso, a estratégia utilizada para a verificação inicial da eficácia da rede foi utilizar a base de dados $fashion\_mnist$ (que é uma base de dados cujo as classes são objetos como tênis, camisetas, calças, etc.) presente no próprio ambiente Keras e a partir dessa base de dados, tomou-se primeiramente as imagens que pertenciam às classes 0 (*T-shirt/top*) e 1 (calças), gerando assim uma nova base de dados que seria utilizada na resolução do problema de classificação binária. Para o conjunto original considerado, há 10 classes possíveis, sendo cada uma composta por 6000 imagens. Portanto, ao separar

*Disponível em: https://github.com/snf/keras-fractalnet/blob/master/src/fractalnet.py

duas classes, obtém-se um conjunto com 12000 imagens, sendo 6000 pertencentes à classe 0 e 6000 pertencentes à classe 1. Como o objetivo era testar o desempenho da rede em problemas de pequeno porte, selecionou-se 3000 imagens de cada classe, totalizando um novo conjunto com 6000 imagens. Assim, considerou-se para o treinamento 4000 imagens e 2000 para a realização da validação. Para os testes, as redes fractais foram testadas com duas configurações diferentes: a primeira, com uma coluna e três blocos e a segunda, com três colunas e três blocos. O objetivo era ver se o aumento da profundidade da rede seria capaz de fornecer melhores respostas. Além desses dois modelos de rede treinados, também foi implementado um modelo de ResNet50[†] e comparado o desempenho deste com as configurações montadas, posteriomente testou-se o desempenho das redes sobre todo o conjunto de 60000 imagens da $fashion\_mnist$, reduzindo o número de épocas.

No artigo que serviu de base para esta pesquisa, o grupo de pesquisadores trabalha sobre o conjunto de dados CIFAR10 e CIFAR100, realizando testes com redes profundas e por um número de épocas relativamente alto (épocas, no contexto de *Deep Learning*, refere-se ao número de vezes que o conjunto de dados de treinamento passa pela rede realizando um processamento completo, ou seja, percorre o sentido convencional e atualiza os parâmetros). No caso deste projeto, a limitação na capacidade de processamento fez com que os testes fossem realizados considerando 10 épocas para o problema de classificação binária e 5 épocas considerando todo o conjunto de dados. Portanto, os resultados obtidos não têm como objetivo verificar a robustez da rede, até porque os autores do projeto já o fazem. O objetivo era verificar o comportamento da rede em problemas de tamanhos distintos e testar se a implementação encontrada realmente fornecia resultados coerentes com a proposta da rede, visto que os autores do projeto utilizaram uma linguagem não muito convencional para a construção da arquitetura e a implementação encontrada em Keras estava disponibilizada em um repositório da plataforma Github, ou seja, seria preciso entender o código e verificar se a implementação realmente fazia sentido.

[†]Disponível em: https://towardsdatascience.com/hitchhikers-guide-to-residual-networks-resnet-in-keras-385ec01ec8ff

5.1 Classificação Binária

A seguir, uma tabela com informações resumidas dos resultados utilizando as duas configurações da rede neural fractal anteriomente mencionadas e da ResNet50 será apresentada e posteriormente haverá a apresentação completa para cada rede de oomo o processo evoluiu ao longo das 10 épocas:

Arquiteturas	Fractalnet (C = 1 e B = 3)	Fractalnet (C = 3 e B = 3)	ResNet50
Acurácia	0.9960	0.9920	0.9530
Custo	0.0125	0.0236	0.1772

A tabela acima reflete os melhoes valores de acurácia na etapa de validação, para cada rede implementada, além do custo associado em cada etapa (valor da função de perda). Nota-se que amabas as redes fractais tiveram um melhor desempenho em comparação à ResNet50, o que fornece indícios de que a rede neural fractal consegue ter bom desempenho mesmo sem a utilização de *Data Augmentation*, ou seja, em um conjunto pequeno de dados disponíveis, mostrou-se superior à rede residual, que por sua vez trabalha muito bem em redes profundas e com grande disponibilidade de dados. Abaixo, seguem as imagens que mostram a evolução da rede por época em cada arquitetura implementada. Observe que ao contrário das redes neurais fractais, a rede residual implementada começa com valores baixos de acurácia e demora mais para convergir. Isto pode ocorrer pela baixa quantidade de exemplos fornecidos e/ou pela complexidade da rede, implicando em um aparente *overfitting* inicial. Por fim, vale comentar que C representa o número de colunas utilizada na rede e B o número de blocos.

Tabela 1: Resultados obtidos utilizando a Fractalnet com C = 1 e B = 3.

Épocas	1	2	3	4	5	6	7	8	9	10
Custo	0.9817	0.0340	0.0237	0.0229	0.0113	0.0196	0.0166	0.0046	0.0039	6.2153e-04
Acurácia (treino)	0.8006	0.9894	0.9926	0.9920	0.9952	0.9923	0.9938	0.9981	0.9987	1.0000
Val_custo	0.4742	0.4091	0.2625	0.1019	0.0549	0.0293	0.0205	0.0148	0.0127	0.0125
Val_acurácia	0.9695	0.9755	0.9760	0.9910	0.9910	0.9915	0.9930	0.9940	0.9945	0.9960

Agora, segue a tabela com os resultados da outra configuração de rede neural fractal montada:

Tabela 2: Resultados obtidos utilizando a Fractalnet com C = 3 e B = 3.

Épocas	1	2	3	4	5	6	7	8	9	10
Custo	1.2838	0.1900	0.1151	0.0507	0.0444	0.0361	0.0411	0.0323	0.0261	0.0242
Acurácia (treino)	0.6522	0.9406	0.9630	0.9831	0.9870	0.9899	0.9845	0.9874	0.9900	0.9899
Val_custo	0.7007	0.7459	0.7269	0.2716	0.0897	0.0833	0.0902	0.0294	0.0254	0.0236
Val_acurácia	0.5000	0.5000	0.5000	0.9850	0.9870	0.9900	0.9900	0.9890	0.9905	0.9920

Como é possível observar, o desempenho ao longo de todo o processo foi extremamente satisfatório. No último caso da rede fractal, com C = B = 3, observa-se que o aumento da profundidade fez com que inicialmente o processo começasse com baixo desempenho, com uma certa tendência à *overfitting* (pela boa acurácia obtida no treinamento), mas rapidamente o processo começa a convergir e apresentar bons resultados. A seguir, é possível notar como o processo evoluiu para o caso da ResNet50:

Tabela 3: Resultados obtidos utilizando a rede residual ResNet50.

Épocas	1	2	3	4	5	6	7	8	9	10
Custo	1.0020	0.1843	0.0928	0.0977	0.0574	0.0418	0.0506	0.0401	0.0339	0.0316
Acurácia (treino)	0.6742	0.9464	0.9709	0.9761	0.9840	0.9895	0.9850	0.9898	0.9921	0.9910
Val_custo	2.7696	4.3032	6.7361	5.4664	6.1326	3.2447	2.3914	0.5737	0.1772	0.5973
Val_acurácia	0.5000	0.5000	0.5000	0.5000	0.5000	0.5005	0.5480	0.8765	0.9530	0.8280

As redes foram treinadas seguindo as seguintes especificações:

1. No caso das três redes, *batch_size* (tamanho do lote a ser enviado por vez para o treinamento em cada época) = 50, épocas = 10, método de otimização = Adam, função de perda = *categorical_crossentropy*.

2. Para as redes neurais fractais: *Momentum* = 0.9, *drop_path* = 0.15 (probabilidade), *dropout*: [0.,0.1,0.2] para C = 1 e B = 3; [0., 0.2, 0.4] para C = B = 3, sendo a posição i (i = 1,2,3) da lista de *dropout* a probabilidade utilizada no bloco i da rede.

3. Adicionalmente na ResNet50, com o objetivo de atenuar possíveis *overfittings* e certa demora na convergência do processo, implementou-se *dropout* também ao longo das camadas, com probabilidade variadas.

5.2 Problema Multiclasses

Nesta etapa, ocorrerá a apresentação dos resultados envolvendo todo o conjunto de dados da $fashion\_mnist$. É importante mencionar que como o ambiente de execução para a realização dos testes foi o *Google Colaboratory*, o processamento ocorria de forma mais demorada e por essa razão os resultados encontrados para 5 épocas descritos a seguir são pouco expressivos em relação à confiabilidade e eficácia das redes. O número de épocas teve que ser diminuído, pois o ambiente de execução reiniciava após um longo tempo e isso fazia com que os cálculos anteriores fossem perdidos. Assim, mesmo com um número praticamente inexpressivo de épocas, o processo todo ainda assim demorou muito para executar todas as 5 requeridas. No caso da ResNet, a operação foi um pouco mais rápida, mas para fins de comparação, também fixou-se 5 épocas para o processo. Além disso, as especificações utilizadas para este caso foram praticamente as mesmas do problema de classificação binária abordado anteriormente, com exceção do *dropout*, que agora foi padronizado em [0., 0.2, 0.4], ou seja, probabilidade 0 para o primeiro bloco, 0.2 para o segundo e 0.4 para o terceiro. A seguir, seguem os resultados obtidos no caso estudado:

Arquiteturas	Fractalnet (C = 1 e B = 3)	Fractalnet (C = 3 e B = 3)	ResNet50
Acurácia	0.9075	0.8306	0.8551
Custo	0.2782	0.4752	0.4203

Assim como no problema de classificação binária, a tabela acima indica os melhores resultados em termos de acurácia ao longo das cinco épocas para cada rede implementada e o custo correspondente, para a etapa de validação. Note que a rede fractal composta por uma coluna e três blocos obteve melhor desempenho em ambos os testes. Segundo Larsson et al. [2017], as redes neurais fractais podem englobar arquiteturas já existentes, ou seja, é possível construir através de configurações determinadas, arquiteturas já existentes e funcionais. De maneira proposital, note que ao implementar uma rede neural fractal com uma coluna e três blocos, o resultado nada mais é do que uma rede neural convolucional com três camadas convolucionais ligadas entre si pela operação de *pooling* logo após o processamento feito em cada uma delas. Com isso, o programa não realiza o *drop_path*, nem o processo de *join*, tornando as operações mais rápidas e podendo assim, justificar um melhor desempenho (pelo menos para poucas épocas) desta

configuração. A seguir, assim como na subseção anterior, haverá a representação do processo ao longo das 5 épocas. Observe que a rede neural fractal com C = B = 3 possui um baixo desempenho logo nas primeiras épocas, enquanto que a ResNet50 aparenta ter uma evolução mais suave no aprendizado. Este fato, mesmo para poucas épocas, pode explicar ou pelo menor fornecer indícios do porquê os autores realizaram testes robustos com cerca de 400 épocas. Provavelmente, além de garantir a convergência do processo e altos níveis de acurácia, garante-se também um resultado livre de *overfitting* (aliado ao emprego de técnicas como *drop_path* e *dropout*).

Tabela 4: Resultados obtidos utilizando a Fractalnet com C = 1 e B = 3.

Épocas	1	2	3	4	5
Custo	0.8566	0.3263	0.2775	0.2496	0.2251
Acurácia (treino)	0.7201	0.8815	0.8975	0.9099	0.9188
Val_custo	0.3941	0.2937	0.2808	0.2782	0.2597
Val_acurácia	0.8745	0.8990	0.9014	0.9075	0.9054

Tabela 5: Resultados obtidos utilizando a Fractalnet com C = 3 e B = 3.

Épocas	1	2	3	4	5
Custo	1.7911	0.7469	0.5452	0.5274	0.4686
Acurácia (treino)	0.4213	0.7283	0.8043	0.8111	0.8273
Val_custo	3.5265	3.3632	0.8711	0.4752	0.5411
Val_acurácia	0.1000	0.1000	0.6922	0.8306	0.8025

Tabela 6: Resultados obtidos utilizando a rede neural residual ResNet50.

Épocas	1	2	3	4	5
Custo	2.0046	0.6874	0.5026	0.4364	0.3846
Acurácia (treino)	0.2912	0.7390	0.8164	0.8386	0.8583
Val_custo	4.5188	0.9446	0.5849	0.5541	0.4203
Val_acurácia	0.2860	0.6718	0.8144	0.8137	0.8551

Nas tabelas acima, em ambas as subseções, Val_custo e Val_acurácia representam o custo e a acurácia da etapa de validação da rede.

6 Conclusão

Ao final deste projeto, é possível concluir que as redes neurais fractais podem vir a ser uma alternativa promissora em relação ao uso das redes neurais residuais, apresentando bom desempenho em problemas de diversos tamanhos e com técnicas inovadoras de regularização. Além disso, possuem flexibilidade na montagem da rede em termos de configurações iniciais, sendo possível estabelecer diferentes níveis de profundidade e com isso aumentar as possibilidades de solução de um dado problema. Entretanto, ressalta-se que os testes realizados neste projeto não são suficientes para verificar a robustez da rede (embora isso tenha sido realizado pelos idealizadores da rede), sendo necessário o emprego da arquitetura em problemas de grande porte. Com isso, caso a eficácia da rede seja realmente comprovada em trabalhos futuros, é possível abrir margem para uma ampla quantidade de pesquisas referentes à como configurar os hiperparâmetros da rede, bem como iniciar a investigação da eficácia dos sub-caminhos da rede, tornando-a ainda mais flexível.

Referências

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *ICLR*, 2017.

Dong Yu Li Deng. *Deep Learning: Methods and Applications*. Foundations and Trends® in Signal Processing 7.3-4. Now Publishers, 2014. ISBN 1601988141, 9781601988140. URL http://gen.lib.rus.ec/book/index.php?md5=d623f585095a41d9ee0c745fe6291fba.